Applied Machine Learning Notebook

Source: https://www.appliedaicourse.com/course/11/Applied-Machine-learning-course

Applied Machine Learning Course

The course that gets you HIRED!

Notebook Written by:

Karthik Kumar Billa [https://www.linkedin.com/in/karthik-kumar-billa/]

Student at Applied AI Course (May 2019 – February 2020)

Completed on: 6th April 2020

Available Modules: 2, 3, 4, 5, 7, 8

Module 1 is just Python introduction, Module 6 and 9 have case studies and live sessions; Case studies and live sessions are not covered in this notebook;

[Google Images]

Chapter 2: Python for Data Science Introduction

2.1 Python, Anaconda and relevant packages

Link: https://www.youtube.com/watch?v=hbUJ6nd-9lA

https://repo.continuum.io/archive/

2.2 Why learn Python?

Python is very simple to pick up;

Packages useful for ML are available in Python;

Jupyter Notebooks for interactive programming;

Extensively used in the industry;

Python is much more general purpose programming language;

2.3 Keywords and identifiers

Keywords are the reserved words in python;

We can't use a keyword as variable name, function name or any other identifier;

Keywords are case sensitive;

[Import keyword]

Example: False, None, True, class, if, else, return, def, try, while, for, etc

Total number of keywords: 33

Identifiers:

Name given to entities like class, functions and variables

Can be a combination of letters, digits and underscores, cannot start with a digit

Keywords cannot be used as identifiers, special characters cannot be used

Python has straight forward Error indications;

2.4 Comments, indentations and statements

Start a line with a # or use triple quotes, ‘’’ ’’’

Indentations are used (4 spaces preferred) to make blocks of code, a for loop

Rather than writing code in a single line try to write in multiple lines (can use \) to make code readable

The written instructions are called statements

2.5 Variables and data types in python

Variable is a location in memory used to store some data; Variable declaration is not needed

a, b = 10, ‘Hi’

id(a) prints location of a

Data types:

Everything in python is an object;

Number: Integers, float and complex

Boolean: True and False

Strings: Sequence of Unicode characters, defined with quotes, indexable, slicable

List: An ordered sequence of items, like an array, can have multiple data type elements, defined with square brackets; Lists are mutable

Tuple: Defined with parenthesis, can have multiple data type elements, tuple is immutable, can be indexable

Set: Defined with Curly braces, Set is an unordered collection of unique items; behaves as a set in mathematics; does not support indexing

Dictionary: an unordered collection of key-value pairs, defined with curly braces and a colon, value accessible with key

Data types can be converted provided the value is valid in both data types;

List(‘Hello’) = [‘H’,’e’,’l’,’l’,’o’]

2.6 Standard input and output

Output: print()

print(‘ {} {}’.format(a,b))

print(‘ {a} {b}’.format(a = 1, b = 2))

Input: input()

2.7 Operators

Operators are special symbols in python that allow arithmetic or logic computation.

2 + 3 : + is an operator and 2, 3 are operands

Types: Arithmetic, Comparison, Logical, Bitwise, Assignment, Special

Arithmetic: +, -, *, /, %, //, ** (addition, subtraction, multiplication, division, modulo division, floor division, exponent)

15, 2: +: 17, - : 13, *: 30, /: 7.5, %: 1 (remainder), //: 7, **: 225;

-15//2 = -8

Comparison: <, >, !=, ==, >=, <=

Logical: and, or, not

Bitwise:

a= 10, b = 4:

a & b: 1010 & 0100: 0000 (and) = 0

a | b: 1110 = 14

or: |, not: -, xor: ~, rightshift: >>, leftshift: <<

a>>b:

Assignment operator:

=, +=, -=, *=, /=, %=, //=, **=, &=, |=

a += 10: a = a + 10

Identity operators:

is, is not

Membership operators:

In, not in

2.8 Control flow: if else

If test expression:

statement(s)

Example:

```
num = 10
if num>0:
    print("number is positive")
elif num ==0:
    print("zero")
else:
    print("number is negative")
print("always printed")
```

2.9 Control flow: while loop

While loop: block of code runs until a test expression is true;

```
lst = [10, 20, 30, 40, 50]
index = 0
while index < len(lst):
    product * = lst[index]
    index += 1 # increment statement is important
```

0: 1*10, 1: 10*20, 2: 200*30,…

We can use an else block when the test condition fails;

2.10 Control flow: for loop

Used to iterate over a sequence;

```
for element in sequence:
    statement(s)
```

```
for ele in lst:
    product *=ele
```

range(): range(10) will generate a list of 10 numbers from 0 to 9;

2.11 Control flow: break and continue

```
for var in sequence:
    if condition:
        break
```

Chapter 3: Python for Data Science: Data Structures

3.1 Lists

Data Structures: collection of data elements

List: Sequence data structures, these are indexable, mutable, defined by square brackets and elements are comma separated

Operations on list:

- len(list), append(element), insert(index, element), remove(element) (removes first occurrence only), list.append(element), list.extend(list), pop(index)
- ['one', 'two'].append(['one', 'two']) = ['one', 'two', ['one', 'two']]
- ['one', 'two'].extend(['one', 'two']) = ['one', 'two', 'one', 'two']
- del lst[1]
- list reverse: list.reverse()
- sorted(list), list.sort()
- lst = [1, 2, 3, 4, 5]; abc = lst; abc.append(6); print(lst) → [1, 2, 3, 4, 5, 6]
  - lst and abc are pointers;
- string.split(' ')
- lst[index]
- lst[slice_index_start: slice_index_end]
- lst1 + lst2
- lst.count()
- for ele in lst: print(ele)
- **List comprehensions:** [i**2 for i in range(10) if i%2 ==0]
  - [[row[i] for row in matrix] for i in range(4)]

3.2 Tuples part 1

A tuple is similar to list; Tuple is immutable, its elements cannot be altered;

- T = "abcd", # comma is important to create a tuple
- Tuple access: T[1]
- Changing a tuple: a list in a tuple is mutable;
- Concat tuples using +

3.3 Tuples part 2

- Deletion: whole tuple will be deleted

- Tuple.count(), tuple.index(element), element in tuple, element not in tuple, len(tuple), sorted(tuple), min(tuple), max(tuple), sum(tuple)

3.4 Sets

Sets are unordered collection of unique items; Mutable, non-indexable;

S = {1, 2, 3}

Sets does not allow duplicate numbers;

set([1,2,3,1]) = (1, 2, 3)

Operations: set.update(elements or sets), set.discard(element), set.remove(element), set.pop(), s.clear()

Set1 | Set2: Union; Set1.union(Set2)

Set1 & Set2, Set1.intersection(Set2)

Set1 – Set2, Set1.difference(Set2)

Set1^Set2, Set1.symmetric_difference(Set2)): Union - Intersection

Frozenset: immutable sets: Set1 = frozenset([1,2,3,4])

3.5 Dictionary

An unordered collection of key value pairs;

O(1) for time complexity for search tasks;

Dictionary is mutable;

Operations: dict.pop(key), dict.clear(), dict.fromkeys(list, values), dict.items(), .keys(), .values(), .copy()

Dictionary Comprehension:

for pair in d.items(): print(pair)

{k:v for k,v in d.items() if v>2}

{k:v for k+’c’,v*2 in d.items() if v>2}

3.6 Strings

Sequence of characters: (Unicode (default) or ASCII)

S = “kl” or = str(1)

Access characters of a string as a list;

Strings are immutable;

Operations: str1+str2; for i in string: _____;

lower(), upper(), join(), split(), find(), replace()

“Bad Morning”.replace(“Bad”, “Good”)

Palindrome:

```
Mystr = “MaDam”
Mystr = Mystr.lower()
revStr = reversed(Mystr)
if list(Mystr) == list(revStr):
        print(“Palindrome”)
else:
        print(“Not palindrome”)
```

Alphabetic sort:

```
Word = word.split().sort()
```

Chapter 4: Python for Data Science: Functions

4.1 Introduction

Functions: a group of related statements that perform a specific task;

Converts a program into smaller chunk which makes management easy

```
def function():
    ‘’’
    Doc string
    ‘’’
    statements
    return
```

Doc strings is written to explain the working of the function (function.__doc__)

Scope and Life Time of Variables: Portion of the code where the variable is recognized and Lifetime is the period throughout which the variable exists in memory

Variable inside a function are local variables which are destroyed once the function finishes execution; Global variables are not destroyed unless deleted;

Program to print highest common factor (HCF):

```
def computeHCF(a, b):
    “””
    Computing HCF of two numbers
    “””
    Smaller =b if a>b else a
    hcf = 1
    for i in range(1, smaller + 1):
```

```
            if (a%i==0) and (b%i==0):
                hcf = i
        return hcf
```

4.2 Types of functions

Built-in Functions and User defined functions;

Built-in: abs(), all(), any(), dir(), divmod(), enumerate(), filter(), map(), reduce(), isinstance(),

enumerate(): returns a list with an index

filter(): applies a function on a list to reduce the list

map(): applies a function on all items of a list

```
    def PowerOfTwo(num):
        Return num**2
    map(PowerOfTwo, list)
```

reduce(): applies a computation and returns a result;

4.3 Function arguments

Functions need inputs which take in values through arguments;

Default arguments: to give default values to a function; have default arguments at the end

Keyword arguments: variable number of arguments can be given as input;

Arbitrary arguments: Used when number of arguments are unknown given as input to the function;

4.4 Recursive functions

Function calling inside itself;

Factorial(n) = n*Factorial(n-1)

Stack of function calls; Makes code clean, hard to debug;

4.5 Lambda functions

Functions without name; used along with filter and map

```
Def Double(x):
    return x*2
```

Can also be defined as:

```
Double = lambda x : x*2
```

Example: list(filter(lambda x : (x%2==0), [1,2,3,4,5])); Output: [2, 4]

reduce(lambda x, y : x* y, [1, 2, 3, 4, 5])

4.6 Modules

Module refers to file containing statements and definitions;

A .py file containing code that is used in other programs

Module: example.py

```
import example
```

- import math
- math.pi

```
import math as m

import datetime
```

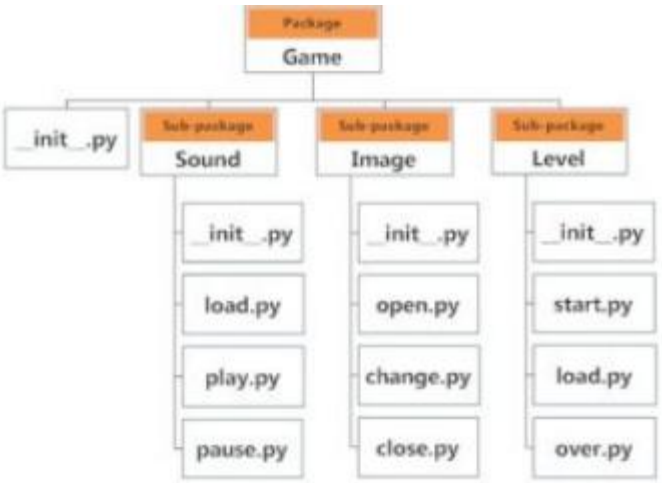
Module is basically a file which contains classes and functions.
Package is a kind of directory which contains modules of similar type.
Library is a collection of Packages.
Framework is collection of Libraries.

Use dir() to get all names functions inside the module

4.7 Packages

__init__.py needs to be present in a folder to consider the folder as a package;

Packages contain modules;



import Game.Sound.play

4.8 File Handling

Storing data in hard disk which is non-volatile:

Open, read and close

File = open('example.txt')

File can be open in different modes:

'r', reading (default)

'w', writing (creates a new file, if exists deletes it)

'a', appending

Closing a file:

File.close() # to make sure the data is not volatile

Use safer "try finally" block: Exception handling

import os

os.mkdir('test')

os.rmdir('test') # does not remove if test is not empty

import shutil

shutil.rmtree('test') # removes non-empty folders

4.9 Exception Handling

Whenever an error is observed, python interpreter raises an error;

Use:

```
import sys

lst = ['b', 0, 2]

for entry in lst:
    try:
        print("The entry is", entry)
        r = 1 / int(entry)
    except:
        print("Oops!", sys.exc_info()[0],"occured.")
        print("Next entry.")
        print("**************************")
print("The reciprocal of", entry, "is", r)
```

```
The entry is b
Oops! <class 'ValueError'> occured.
Next entry.
```

Raising exceptions:

For example: Memory error

```
try:
    f = open('sample.txt')
    #perform file operations

finally:
    f.close()
```

Finally block runs at the end of all operations, such as closing the file even if the file is not written, to save data.

4.10 Debugging Python

Python debugger: pdb

```
def seq(n):
    for i in range(n):
        print(i)
    return

seq(5)
```

```
0
1
2
3
4
```

```
import pdb

#interactive debugging
def seq(n):
    for i in range(n):
        pdb.set_trace() # breakpoint
        print(i)
    return

seq(5)
```

Comes to pdb.set.trace at every iteration, requires string input to give results

Module 2: Data Science: Exploratory Data Analysis and Data Visualization

Chapter 10: Plotting for exploratory data analysis (EDA)

10.1 Introduction to IRIS dataset and 2D scatter plot

EDA: Simple analysis to understand data; tools include statistics, linear algebra and plotting tools

IRIS dataset: Hello World of Data Science; Collected in 1936; Classify a flower into 3 classes;

Features: Sepal Length, Sepal Width, Petal length, Petal Width

Columns: Variables or features

Rows: Data points or vector: n-dimensional numerical variable

Fisher's *Iris* Data

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	*I. setosa*
4.9	3.0	1.4	0.2	*I. setosa*
4.7	3.2	1.3	0.2	*I. setosa*
4.6	3.1	1.5	0.2	*I. setosa*
5.0	3.6	1.4	0.3	*I. setosa*
5.4	3.9	1.7	0.4	*I. setosa*
4.6	3.4	1.4	0.3	*I. setosa*
5.0	3.4	1.5	0.2	*I. setosa*

Species: Target/ Label; A 1D vector: Scalar;

These features are determined through domain knowledge

Q: Number of data points in the data set? .shape

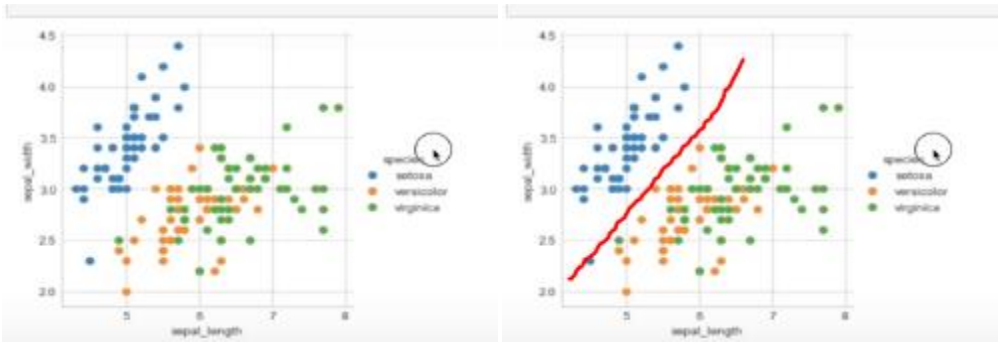
Analysis is more important than code;

Q: What are the features? .columns

Q: Number of points in each class? .value_counts() (Also gives idea of the balance in class labels across number of data points)

Q: 2D scatter plot? (iris.plot(kind = 'scatter', x = 'sepal_length', y='sepal_width')

Use seaborn to separate different classes on above plots

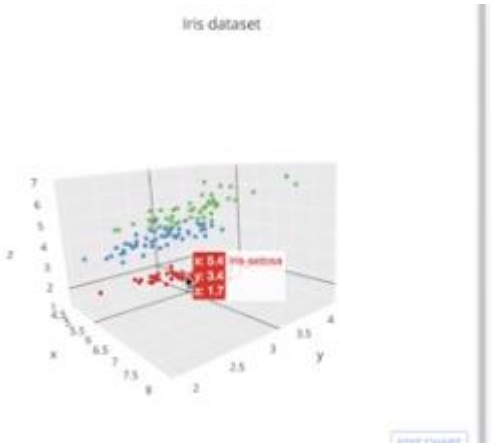


One of the labels can be separated easily by drawing a line;

EDA gives us these neat characteristics of the data set;

10.2 3D scatter plots

Use plotly;



We cannot visualize 4D, 5D or nD (requires mathematical tools to reduce complexity)

10.3 Pair plots

To allow visualization upto 6D sometimes upto 10D, Larger nD cannot be visualized;

Seaborn.paiplot(iris, hue = "species")

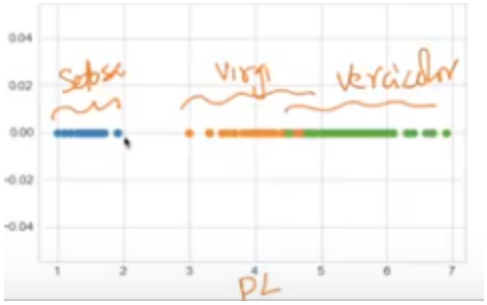
We can write straight forward rules (if else) to separate one of the class labels;

10.4 Limitations of Pair plots

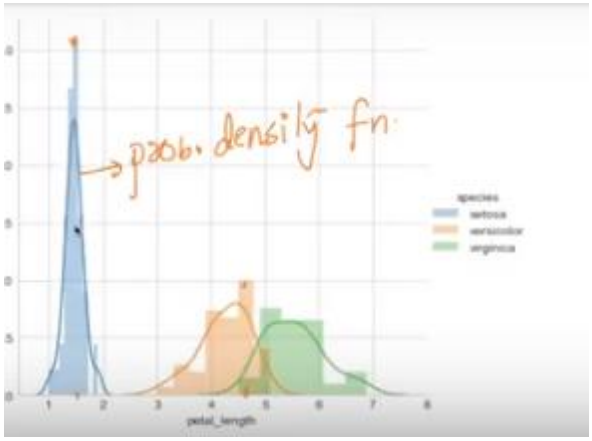
Pair plots are useful when the dimensionality is small

10.5 Histogram and Introduction to PDF (Probability Density Function)

Uni-Variate analysis:



Histograms:

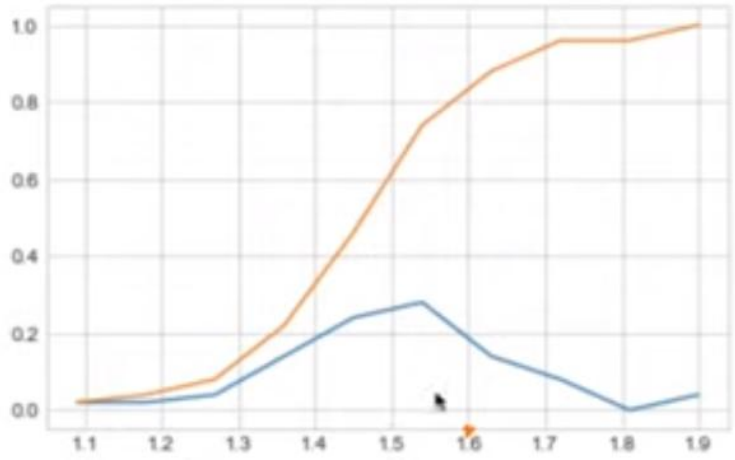


PDF: smoothed histogram (Kernel Density Estimation)

10.6 Univariate Analysis using PDF

Histograms plot on each variable (as the number of features is less, = 4)

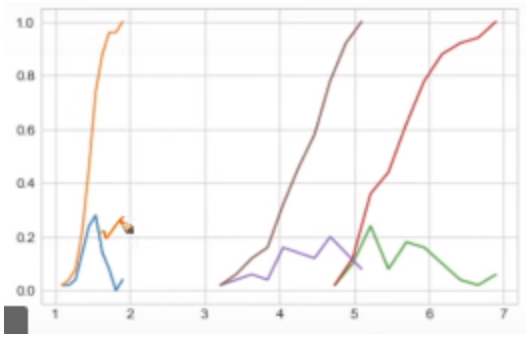
10.7 CDF (Cumulative Distribution Function)



CDF: Percentage of data points that are below an x value;

PDF: Percentage of data points that lie between two x values;

Differentiate cdf to get pdf and integrate pdf to get cdf; Convert data into bins to get pdf, and use npp.cumsum to get cdf;



We can get intersections of CDF levels on x axis above to know how many points of each class intersect

10.8 Mean, Variance and Standard deviation

Mean: Sum(data values)/number of values

central value of the data, are majorly impacted by outliers

Standard deviation: The width of the spread of the values on an axis

Variance: Summation squares of distance of values from mean whole divided by N

Standard deviation = sqrt(variance)

These are corrupted if an outlier exists; use median

10.9 Median

Median does not get corrupted due to presence of outliers; better statistic for central tendency;

Sort the list in an increasing order and pick the middle value;

10.10 Percentile and Quantiles

Percentile: Number of values lesser than the percentile percentage of values;

50^{th} percentile: the value at which 50% of values are less than this value

Quantiles: 25^{th} percentile, 50^{th} percentile, 75^{th} percentile

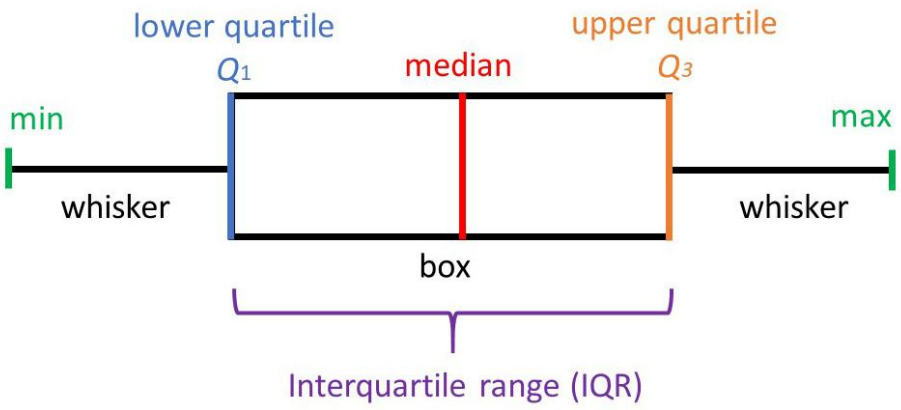
10.11 IQR (Inter Quartile Range) and MAD (Median Absolute Deviation)

Median Absolute Deviation: (1/n) * sum(abs($value_i$ – median))

IQR: 75^{th} percentile value – 25^{th} percentile value

10.12 Box – plot with Whiskers

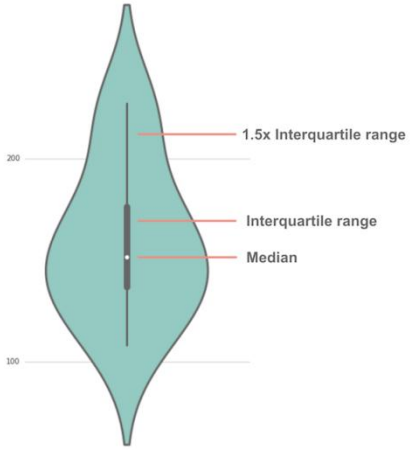
The **box plot** (a.k.a. **box and whisker diagram**) is a standardized way of displaying the distribution of data based on the five number summaries: minimum, first quartile, median, third quartile, and maximum. (Google Search)



10.13 Violin Plots

A **violin plot** is a method of **plotting** numeric data. It is similar to a **box plot**, with the addition of a rotated kernel density **plot** on each side. **Violin plots** are similar to **box**

plots, except that they also show the probability density of the data at different values, usually smoothed by a kernel density estimator.



[Google Search]

10.14 Summarizing Plots, Univariate, Bivariate and Multivariate analysis

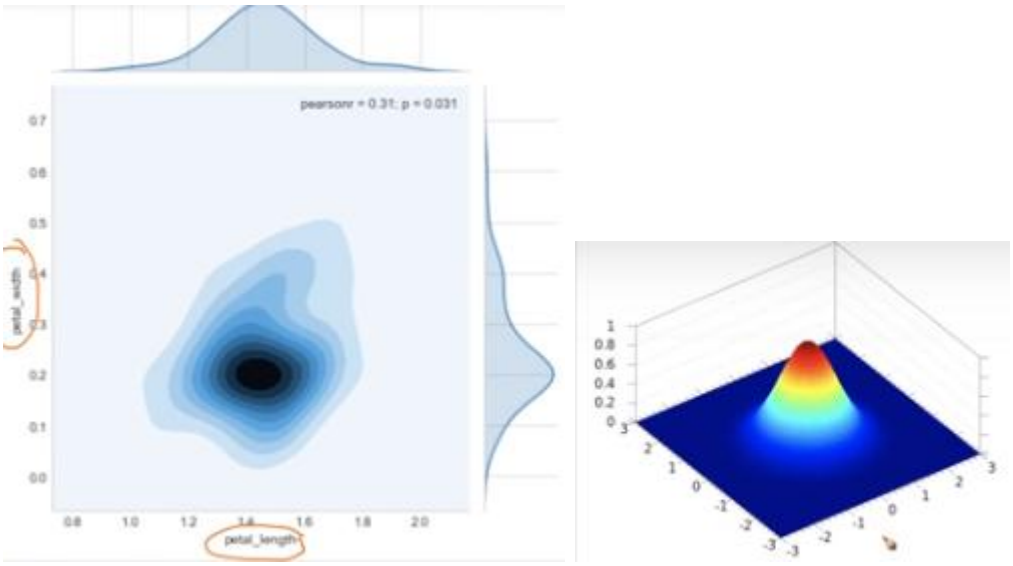
Write conclusion at end of each step or plots during data analysis, data analysis should align with project objective;

Univariate: Analysis considering only one variable (PDF, CDF, Box-plot, Violin plots)

Bivariate: two variables (Pair plots, scatter plots)

Multivariate: more than two variables (3D plots)

10.15 Multivariate Probability Density, Contour Plot



Combining probability densities of two variables; dense regions are darker as if a hill coming out

Chapter 11: Linear Algebra

11.1 Why Learn it?

We will apply it to solve specific type of problems in ML; we will learn things in 2d and 3d and extend it to nd

11.2 Introduction to Vectors (2-D, 3-D, n-D), Row Vector and Column Vector

Point:

Let us have a Coordinate system with x1 and x2 axes;

Point can be denoted by n-dimensional vector;

Distance of a point from origin:

In 2D: $\text{sqrt}(x1^2 + x2^2)$

In 3D: $\text{sqrt}(x1^2 + x2^2 + x3^2)$

In nD: $\text{sqrt}(x1^2 + x2^2 + x3^2 + \ldots. + xn^2)$

Distance between two points:

In 2D: $\text{sqrt}((x1-y1)^2 + (x2-y2)^2)$

In 2D: $\text{sqrt}((x1-y1)^2 + (x2-y2)^2 + (x3-y3)^2)$

In 2D: $\text{sqrt}((x1-y1)^2 + (x2-y2)^2 + \ldots + (xn-yn)^2)$

Row vector: A matrix with n columns and 1 row

Column Vector: A matrix with 1 column and n rows

11.3 Dot Product and Angle between 2 Vectors

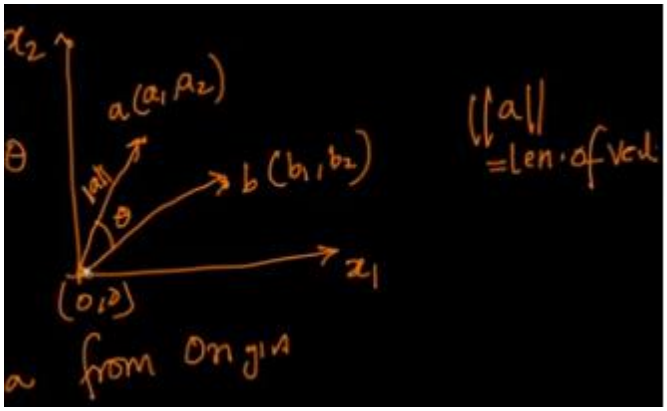
a = [a1, a2, ……., an]

b = [b1, b2, ……, bn]

c = a + b = [a1 + b1, a2 + b2, ….., an + bn]

Dot product: a.b = a1b1 + a2b2 + ….. anbn = a^T * b (by default every vector is a column vector)

Geometrically represents the angle between the two vectors;



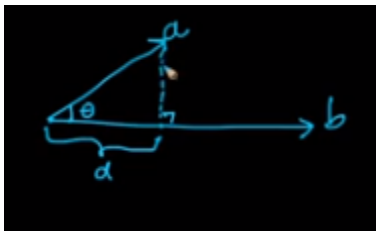
$a.b = ||a||\ ||b||\ \cos(\theta_{ab}) = a1b1 + a2b2 + \ldots..$

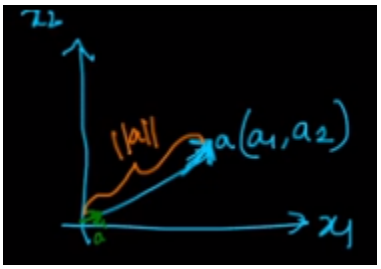
$||a|| = \text{sqrt}\ (a1^2 + a2^2 + \ldots..)$

If the dot product of two vectors is zero then the two vectors are orthogonal.

11.4 Projection and Unit Vector

Projection of a on b = d = $a \cos(\theta_{ab}) = a.b/||b||$





Unit Vector: $\hat{a} = a/||a||$

Unit vector â is in same direction as a; $||\hat{a}|| = 1$

11.5 Equation of a line (2-D), Plane (3-D) and Hyperplane (n-D), Plane passing through origin, Normal to a Plane

Line 2D: $y = mx + c$, $ax + by + c = 0$, $y = -c/b - ax/b$

Plane 3D: $ax + by + cz + d = 0$

Hyper Plane nD: $w0 + w1x1 + w2x2 + w3x3 + \ldots.. = 0$

Hyper plane π: $W^T x + W0 = 0$

$W_{n+1x1} = [W1, W2, \ldots.]$; $X_{n+1\ x\ 1} = [x1, x2, \ldots]$ (W0 determines intercept of the hyper plane on y – axis] (To pass through origin this intercept must be 0)

π: $W^T x = 0$: Hyper plane passing through origin

$W.x = W^T x = ||w||\ ||x||\ \cos(\theta_{xy})$

$\theta_{xy} = 90^0$; W is vector perpendicular to the plane π

11.6 Distance of a point from a plane/ Hyperplane, half-spaces

π: $W^T x = 0$: Hyper plane passing through origin

$\theta_{xy} = 90^0$; W is vector perpendicular to the plane π

Distance of a point P (p1, p2, …):

$$d = abs(W^T P / ||W||)$$

A hyper plane divides the whole space into two half spaces;

If W.P > 0 then point P lies in the direction of W;

11.7 Equation of a Circle (2-D), Sphere (3-D) and Hypersphere (n-D)

Circle: $(x-a)^2 + (y-b)^2 = c^2$

Given a point: if its distance from the center is less than the radius of a circle then the point lies inside the circle;

$x_1^2 + x_2^2 < r^2$ then the point p(x1,x2) lies inside the circle that is centered at origin;

$x_1^2 + x_2^2 > r^2$ then the point p(x1,x2) lies outside the circle;

$x_1^2 + x_2^2 = r^2$ then the point p(x1,x2) lies on the circle;

Sphere: $x_1^2 + x_2^2 + x_3^2 = r^2$

Hyper sphere: $x_1^2 + x_2^2 + x_3^2 + \ldots\ldots + x_n^2 = r^2$

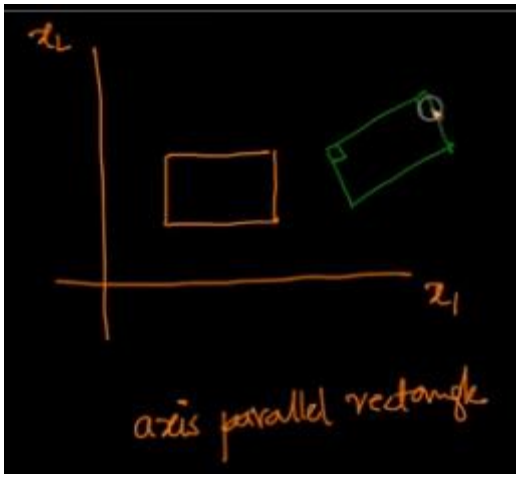
11.8 Equation of an Ellipse, Ellipsoid and Hyper ellipsoid

Ellipse: $(x/a)^2 + (y/b)^2 = 1$

Ellipsoid: $(x_1/a)^2 + (x_2/b)^2 + (x_3/c)^2 = 1$

Hyper sphere: $(x_1/a_1)^2 + (x_2/a_2)^2 + (x_3/a_3)^2 + \ldots\ldots + (x_n/a_n)^2 = 1$

11.9 Square, Rectangle



if x1 < a2 and x1>a1: if y1>b1 and y1<b2 then point P (x1, y1) lies inside the axis parallel rectangle, if a2 – a1 = b2 – b1 then we get a square

Can be extended to 3D

11.10 Hyper Cube, Hyper Cuboid

if x < a2 and x>a1: if y>b1 and y<b2: if z>c1 and z<c2 then point P (x, y,z) lies inside the axis parallel cuboid, if a2 – a1= b2 – b1 = c2 – c1 then we get a cube;

11.11 Revision Questions

Q1. Define Point/Vector (2-D, 3-D, n-D)?

Q2. How to calculate Dot product and angle between 2 vectors?

Q3. Define Projection, unit vector?

Q4. Equation of a line (2-D), plane(3-D) and hyperplane (n-D)?

Q5. Distance of a point from a plane/hyperplane, half-spaces?

Q6. Equation of a circle (2-D), sphere (3-D) and hypersphere (n-D)?

Q7. Equation of an ellipse (2-D), ellipsoid (3-D) and hyperellipsoid (n-D)?

Q8. Square, Rectangle, Hyper-cube and Hyper-cuboid?

Chapter 12: Probability and Statistics

12.1 Introduction to Probability and Statistics

Datasets with well separable classes do not require ML methods to be applied, but classes that are well mixed cannot be separated easily;

The data points that lie in the mixed region cannot be clearly stated to belong to a certain class; instead we can have probability values of the data point belonging to each class;

Histograms, PDF, CDF, mean, variance, etc come under probability and statistics

Random Variable: can take multiple values in a random manner

Ex: Dice with 6 sides: Roll a fair dice: the 6 outcomes of the dice are equally likely

r.v. X = {1, 2, 3, 4, 5, 6}

Coin toss: r.v. X = {H, T}

Dice roll: Probability of X = 1 is 1/6

Probability of X = even is {2,3,4}/{1,2,3,4,5,6} = 3/6 = ½

= prob(x=2) + prob(x=4) + prob(x=6)

Height of a randomly picked student: Y = [120 to 190 cm]

Discrete random variable: a value from a set of values

Continuous random variable: a value from a range of values

Outlier: Y: Height of students:

Let: Y = {122.2, 146.4, 132.5, …, 12.26, 156.23, 92.6}

Outliers: 92.6 and 12.26: may have occurred due to input error or data collection error or may be a genuine value but does not indicate general trend of the population

12.2 Population and Sample

Population example: Set of heights of all people in the world;

Task: estimate the average height of a human;

Mean = (1/n) sum(h)

Collecting all the population height data is not possible; we will take a random sample (subset of the population)

12.3 Gaussian/Normal Distribution and its PDF (Probability Density Function)

Bell shaped curve: Gaussian distribution Probability density function curve;

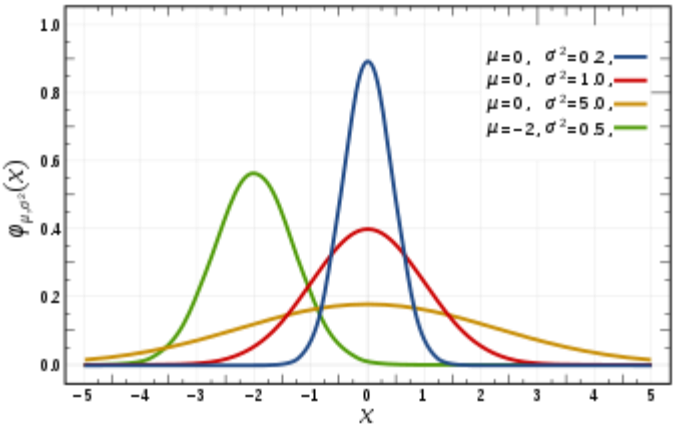
X: continuous random variable;

Real world variables mostly follow Gaussian distribution; Height, Weight, Length

We learn distributions to understand the underlying statistics or trends of the population

If X behaves as a Gaussian distribution: Given mean and standard variance:

We can estimate the values of the population; by plotting the PDF



The red curve is standard normal distribution

Parameters of Gaussian distribution: Mean and variance:

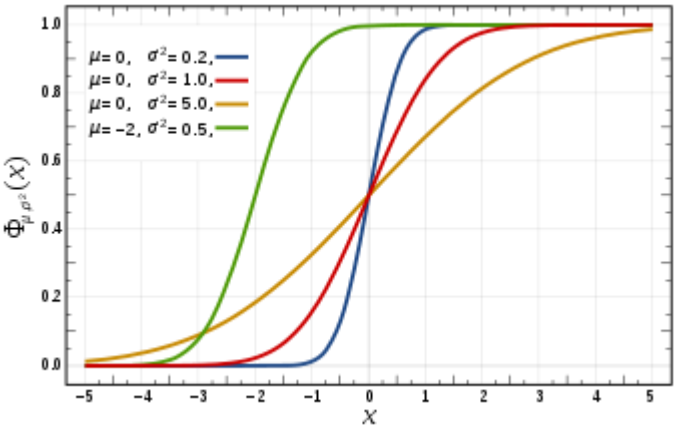
X ~ N (0,1) (mean 0 and variance 1)

PDF(X = x) = $(1/\text{sqrt}(2\pi)\ \sigma)\ \exp(-(x-\mu)^2/2\sigma^2)$

As x moves from mean, the pdf reduces;

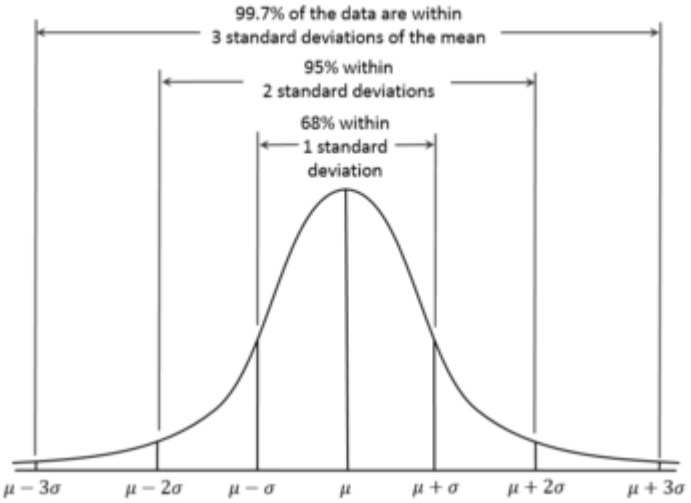
The PDF of a Normal distribution is symmetric;

12.4 CDF (Cumulative Distribution Function) of Gaussian/Normal distribution



$N(\mu,2\sigma^2)$;

As variance decreases the plot gets converted to a step function;



Standard deviation plot: 68 – 95 – 99.7 rule

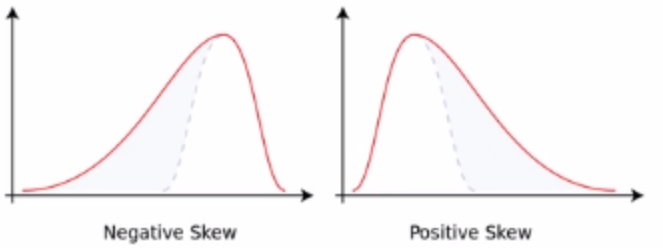
12.5 Symmetric distribution, Skewness and Kurtosis

One of the applications of these stats is to understand the distribution;

If the PDF has a mirror image of the curve of the either sides of the mean, then the distribution is symmetric over mean; as in above std dev plot;

$F(x_0 - h) = F(x_0 + h)$: function symmetric over x_0

Skewness: around the mean the distribution is not symmetric; one of the sides has a longer tail; can be said to be a measure of asymmetry;



Sample Skewness:

$$b_1 = \frac{m_3}{s^3} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2}^3} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^3}{\left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2\right]^{3/2}},$$

where $\overline{x}$ is the sample mean, s is the sample standard deviation, and the numerator m_3 is the sample third central moment.

Kurtosis:

Excess kurtosis: kurtosis – 3

$$g_2 = \frac{m_4}{m_2^2} - 3 = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2\right)^2} - 3$$

where m_4 is the fourth sample moment about the mean, m_2 is the second sample moment about the mean (that is, the sample variance), x_i is the i^{th} value, and $\overline{x}$ is the sample mean.

Kurtosis of Gaussian random variable =3;

Through excess kurtosis we compare distributions with Gaussian random distribution

Kurtosis is a measure of tailedness; it is not a measure of peakedness;

This will give us an idea of outliers in the distribution; Smaller the better

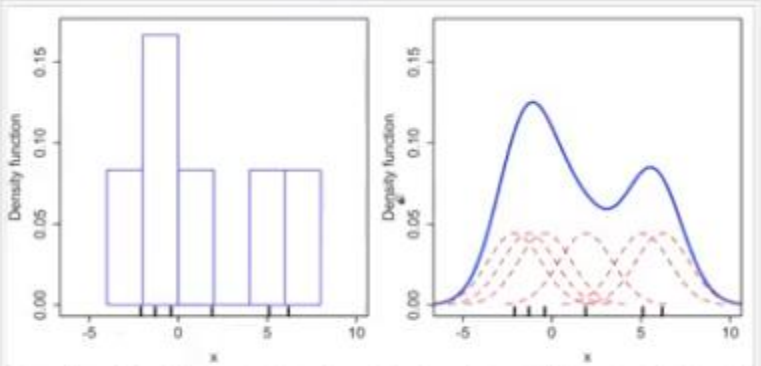
12.6 Standard normal variate (Z) and standardization

Z ~ N(0,1), mean = 0 and variance = 1

Let X ~ N(μ,σ^2); this can be converted to Z by:

Mean centering and scaling (X – mean) / std dev; To help understand the disb.

12.7 Kernel density estimation



Comparison of the histogram (left) and kernel density estimate (right) constructed using the same data. The 6 individual kernels are the red dashed curves, the kernel density estimate the blue curves. The data points are the rug plot on the horizontal axis.

Computing PDF from Histograms: using KDE;

A Gaussian Kernel is plotted centered at each observation (of the histogram);

Variance of this Kernel (bell curve) is called band width;

At every point in the range of the Gaussian Kernels we will add all PDF values to get a combined PDF; bandwidth selection is done by experts;

12.8 Sampling distribution & Central Limit theorem

Let X be a not necessarily Gaussian distribution of a population (say incomes)

Pick m random Samples independently of size **n** each → S1, S2, S3, …., Sm

For all Samples: compute means, x1m, x2m, x3m, ……

These sample means will have a distribution: Sampling distribution of sample-mean

Central Limit Theorem: If X (original population), X: finite mean and variance

Sampling distribution of sample means will be a Gaussian distribution with
mean = population mean and variance = population variance/**n** as n increases

It is generally observed that if n is around 30 and m = 1000, we can observe a Gaussian distribution;

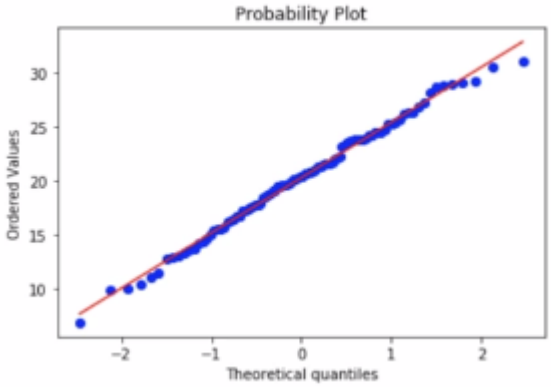
12.9 Q-Q plot: How to test if a random variable is normally distributed or not?

Quantile – Quantile plot: Graphical way of determining similarity between two distributions

X: a new distribution: Task: to understand whether X is Gaussian

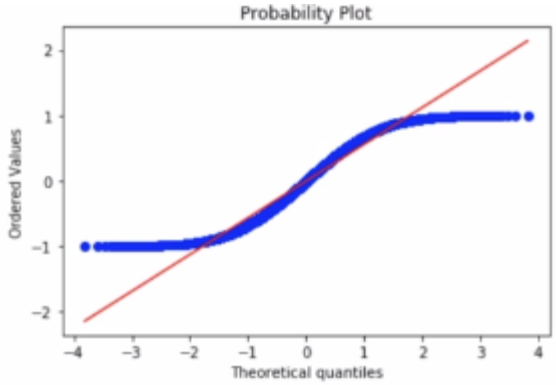
1. Sort X values and compute percentiles;
2. Y ~ N(0,1): take some k observations and sort + compute percentiles
3. Plot: X percentile values on Y axis and Y percentile values on x axis

If the plot is approximately a straight line then the distributions are similar;



Code: stats.probplot(X, dist = 'norm', plot = pylab)

If number of observations is small it is hard to interpret QQ plot;



Plot where distributions are different;

12.10 How distributions are used?

Probability is useful for data analysis;

Ex: Imagine your task is to order T shirts for all employees at your company. Let sizes be S, M, L and XL. Say we have 100k employees who have different size requirements;

Q) How many XL T shirts should be ordered?

Collect data for all 100k employees

Let us have a relationship between heights and T shirt size; Domain knowledge;

Collect heights from 500 random employees; Compute mean and std dev

At gate of entry we can do this;

From domain knowledge, let heights ~N(mean, variance)

We can extend the distribution of heights from 500 employees to 100k employees; We made many assumptions here; these may work in natural features

Q) Salaries: If salaries are Gaussian distributed, we can estimate how many employees make a salary > 100k $;

So distributions give us a theoretical model of a random variable; This will help us understand the properties of the random variable;

12.11 Chebyshev's inequality

If we don't know the distribution and mean is finite and standard deviation is non-zero and finite;

Task: to find the percentage of values of lie within a range;

Salaries of individuals (millions of values); distribution is unknown, but mean and std dev are known;

$P(|x - \mu| >= k\sigma) <= 1/k^2$

12.12 Discrete and Continuous Uniform distributions

PDF for continuous and PMF for discrete random variables;

Roll a dice: uniform distribution; each observation is equally probable;

Discrete Uniform:

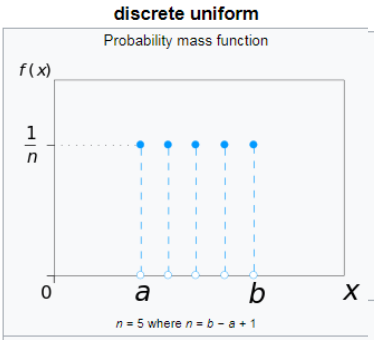
Parameters (a,b,n=b-a+1)

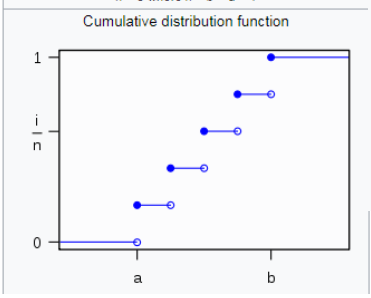
Pmf = 1/n

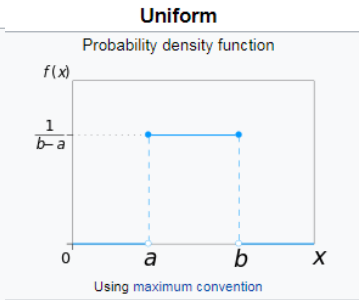
Mean = (a+b) /2 = Median

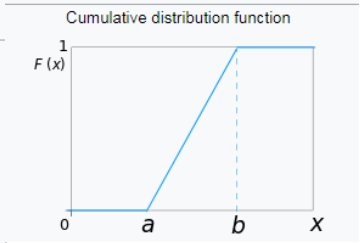
Variance = $((b-a+1)^2-1)/12$

Skewness = 0









Continuous Uniform:

Parameter: a,b

PDF: 1/(b-a)

Mean: (a+b)/2 = Median

Variance = $(b-a)^2/12$

12.13 How to randomly sample data points (Uniform Distribution)

Using random functions in python;

We can use a threshold value to generate a random number of random numbers;

Probability of picking any value is equally probable;

12.14 Bernoulli and Binomial Distribution

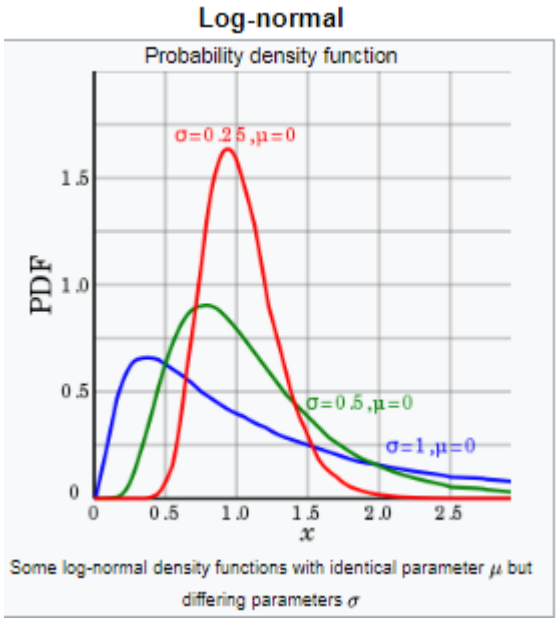
Bernoulli: discrete: Coin toss:{H, T}, P = {0.5, 0.5}; parameter p

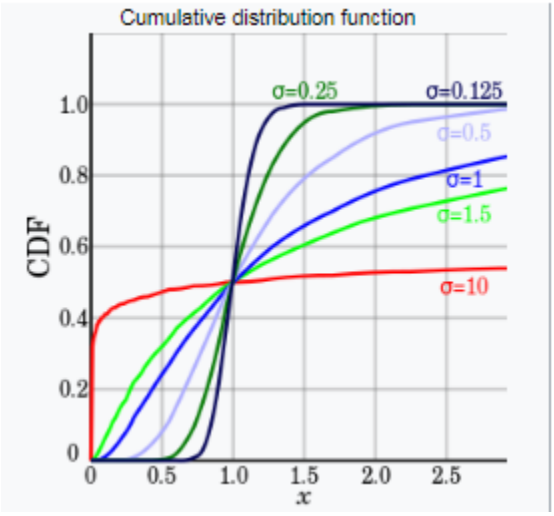
Binomial: X ~ Bernoulli and Y = n times X; parameters: n, p

An event with n trials with success probability p in each trial and p = 0 or 1

12.15 Log Normal Distribution

X is log normal if natural logarithm of x is normally distributed;

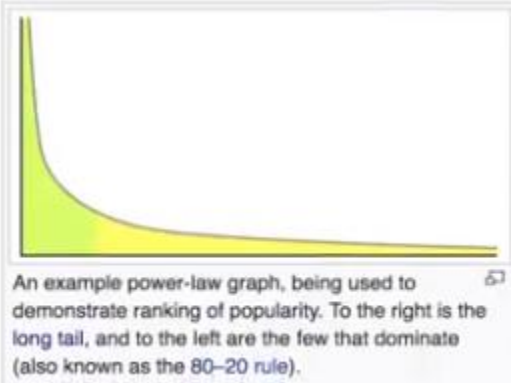




In user reviews: most of comments are of small length and some of comments have large length of words;

We can employ QQ plot to check similarity of log of distribution to normal distribution;

12.16 Power law distribution

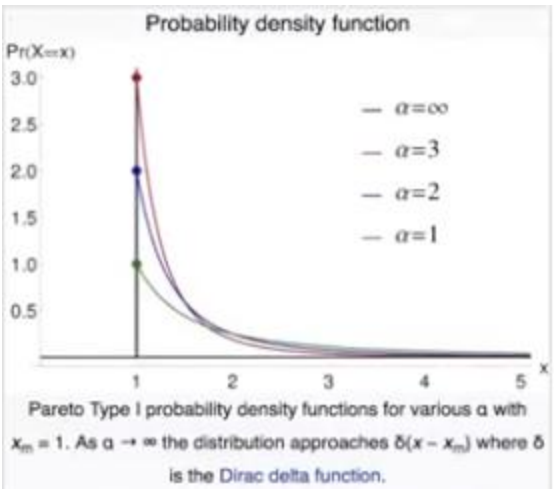


Green area is 80% values;

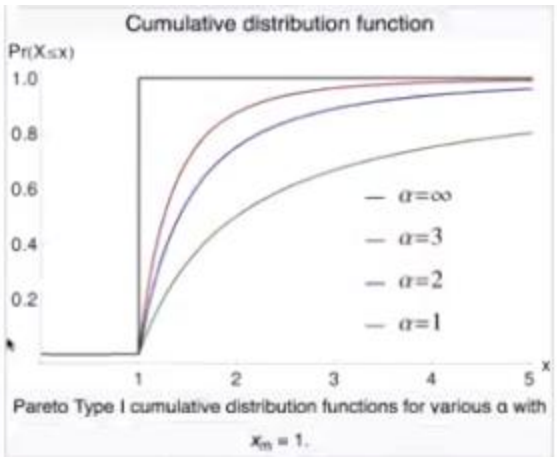
In bottom 20% of values you can find 80% of mass;

When a distribution follows a power law then the distribution is called Pareto distribution;

Parameters: scale and shape

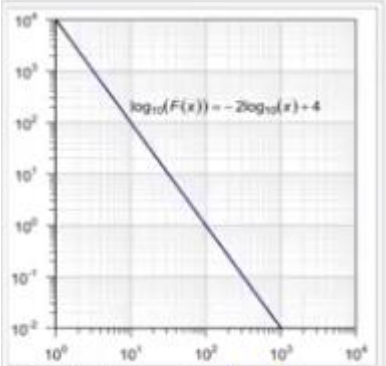


Pareto Type I probability density functions for various α with $x_m = 1$. As $\alpha \to \infty$ the distribution approaches $\delta(x - x_m)$ where δ is the Dirac delta function.



Pareto Type I cumulative distribution functions for various α with $x_m = 1$.

Applications: allocation of wealth in population, sizes of human settlements;



A straight line on a log–log plot is strong evidence for power-laws, the slope of the straight line corresponds to the power law exponent.

We can also use a QQ plot against Pareto plot;

12.17 Box cox transform

Power transform:

In Machine Learning we assume generally that a feature follows a Gaussian distribution;

Can we convert Pareto into Gaussian?

Pareto: X: x1, x2, …., xn & Gaussian Y: y1, y2, ….., yn

Box-cox(x) = λ

$y_i = \{(x_i^{\lambda} - 1) / \lambda$ if λ!=0 and $\lg(x_i)$ if $\lambda = 0\}$

If λ = 0, then the power law distribution is actually a log normal;

scipy.stats.boxcox()

Box-cox does not work always. Use QQ-plot to check its results.

12.18 Applications of non-Gaussian distributions?

Uniform for generating random numbers;

A well studied distribution gives a theoretical model for the behavior of a random variable

Weibull distributions:

The upstream rainfall determines the height of a dam which stands for 100s of years without repairs; Probability of rainfall > a value is required;

This distribution is applied to extreme events such as annual maximum one day rainfalls and river discharges.

The probability density function of a Weibull random variable is:[1]

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & x \geq 0, \\ 0 & x < 0, \end{cases}$$

where $k > 0$ is the *shape parameter* and $\lambda > 0$ is the *scale parameter* of the distribution.

12.19 Co-variance

Task: To determine relationships between different random variables;

Cov(x, y) = (1/n) * sum($(x_i - \mu_x)$ $(y_i - \mu_y)$)

Cov(x, x) = var(x)

Cov(x, y) will be positive if as x increases y increases

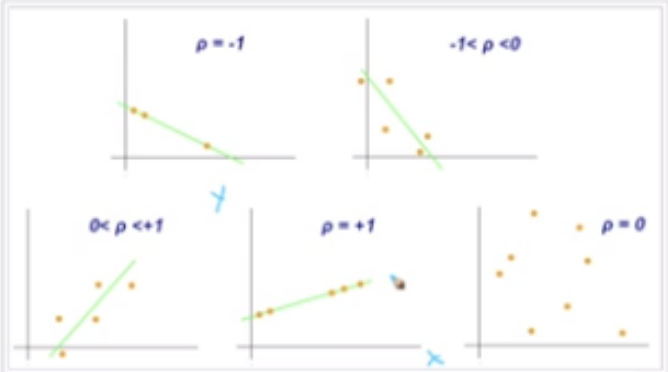
Cov(x, y) will be negative if as x increases y decreases

Sign of co-variance is indicative of relationship, the value of co-variance does not indicate strength of relationship;

12.20 Pearson Correlation Coefficient

To measure relationship between two random variables: ρ

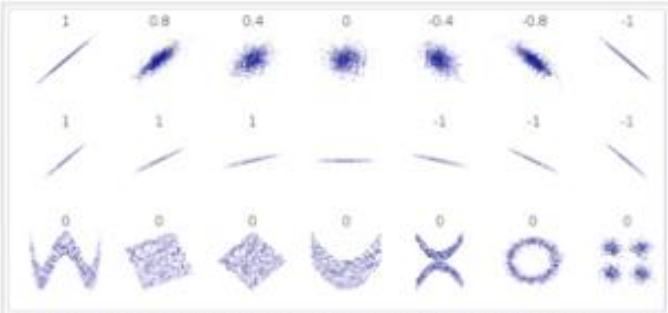
$P_{x,y} = Cov(x, y) /(\sigma_x, \sigma_y)$



Examples of scatter diagrams with different values of correlation coefficient (ρ)

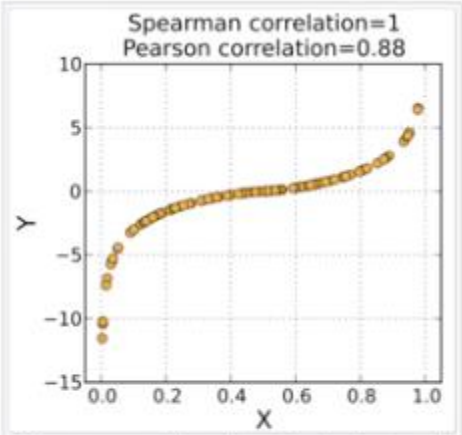
ρ = 0 → no relation between the two random variables;

Pearson relation coefficient assumes that the relationship is linear:



Several sets of (*x*, *y*) points, with the correlation coefficient of *x* and *y* for each set. Note that the correlation reflects the non-linearity and direction of a linear relationship (top row), but not the slope of that relationship (middle), nor many aspects of nonlinear relationships (bottom). N.B.: the figure in the center has a slope of 0 but in that case the correlation coefficient is undefined because the variance of *Y* is zero.

Does not perform well with non-linear relationships;



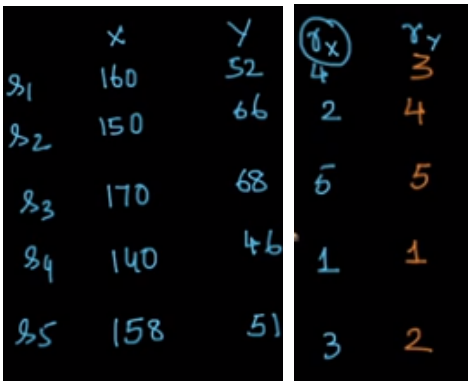
A Spearman correlation of 1 results when the two variables being compared are monotonically related, even if their relationship is not linear.

The above plot has a monotonically non decreasing curve

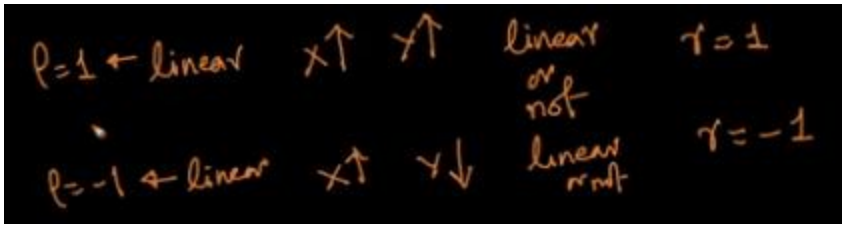
12.21 Spearman Rank Correlation Coefficient

Pearson corr coeff: works for linear relationships

Spearman rank corr coeff: this is Pearson correlation coefficient of ranks of the random variables; Spearman correlation coefficient is much more robust to outliers than Pearson Corr coef



This allows us to understand whether two random variables increase at the same time;



Pearson Spearman

12.22 Correlation vs Causation

Correlation does not imply causation;

Example: Number of Nobel Laureates vs Chocolate consumption per capita;

The rank correlation coefficient ~ 0.8

As x increased y increased;

But we know that as chocolate consumption and Number of Nobel Laureates do not relate;

Causations are studied with causal models which is a separate field of study;

12.23 How to use correlations?

Q. Is salary correlated with sq footage of home?

This data will be useful for real estate agents;

Q. Is # of years of education correlated with income?

Useful for education ministry to encourage people to get more years of study;

Q. Is time spent on web page in last 24 hours correlated with money spent in the next 24 hours?

Useful for ecommerce to encourage people to spend more time on the website;

Q. Is number of unique visitors to the website correlated with the $ sales in a day?

The company will then take measures to increase number of unique user in a day;

Q. Dosage correlation with reduction in blood sugar;

Correlations can help us answer these questions;

12.24 Confidence interval (C.I) Introduction

Let X be a random variable of height values;

Distribution of X is unknown;

Task: Estimate the population mean of X = μ

$\mu = (1/n) * \text{summ}(i = 1 \text{ to } n) (x_i)$

As n increases sample mean reaches population mean;

With μ we have a point estimate and μ is sample mean that is calculated over a randomly selected sample of the population;

Instead we can also express mean in terms of confidence interval;

Say we have 170 cm as sample mean; we can say that sampling mean lies between 165 and 175 cm for 95% of the sampling experiments when the repeat the sampling multiple times; for 5% of the experiments the mean will not fall in this interval; This does not mean that population mean will lie in the interval with 95% probability;

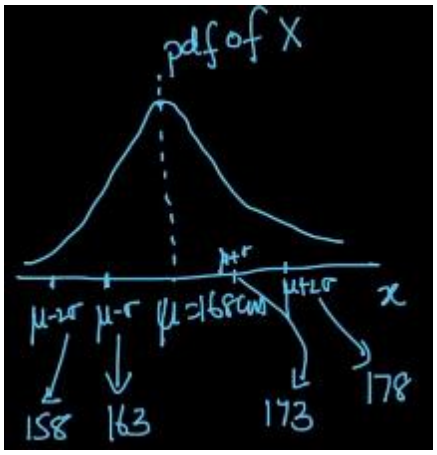
Confidence Interval: An N% confidence interval for parameter p is an interval that includes p with probability N%; [Source: Tom Mitchell book]

12.25 Computing confidence interval given the underlying distribution

Imagine we have a random variable of heights of population;

Let this random variable follow a Gaussian distribution;

Let mean = 168 cm and standard deviation = 5



We can get 95% CI from Gaussian distribution characteristics which range from $\mu - 2\sigma$ to $\mu + 2\sigma$, which is 158 to 178 cm;

12.26 C.I for mean of a random variable

Random variable $X \sim F(\mu, \sigma)$

Let sample size be 10;

Let samples be {180, 162, 158, 172, 168, 150, 171, 183, 165, 176}

What is the 95% CI for population mean; the distribution can be anything;

Case 1: Let σ = 5cm;

CLT: sample mean x_bar follows Gaussian distribution with mean = population mean and standard deviation = population standard deviation/sqrt(sample size);

We can say that μ ε [x_bar - 2σ/sqrt(n), x_bar + 2σ/sqrt(n)] with 95% confidence;

Thus 95% CI for the samples:

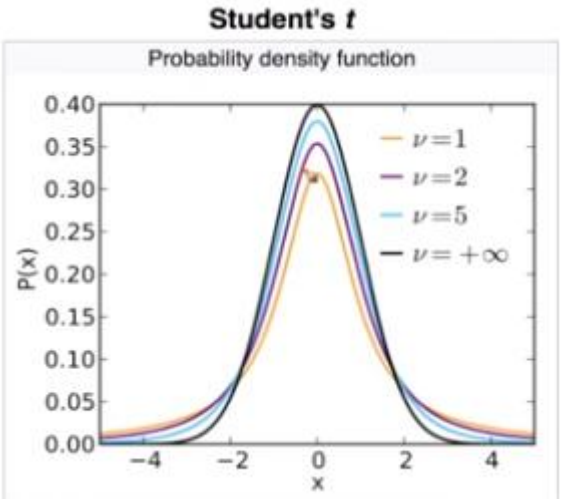
x_bar = 168.5

n = 10

Population Mean lies between 165.34 and 171.66 with 95% confidence;

Case 2: if σ is unavailable: CLT cannot be applied;

We can assume Student's t distribution;



ν = sample size – 1

We can apply 95% characteristics of t distribution;

We have CI for mean; what if we like to have CI for standard deviation, median, 90th percentile and other statistics;

12.27 Confidence interval using bootstrapping

Let X be a random variable with unknown distribution F;

Let us compute confidence interval for median of X;

S: sample size n {x1, x2, …, xn} n = 10, 20, 40 or 50

From sample we generate new samples:

S1: x1(1), x2(1), …., xm(1) and m<=n;

Random sample with replacement of size m generated from sample S;

Using S as a uniform random variable;

Generate k samples;

K samples generated from Sample S with replacement;

For each sample we can generate the statistic desired;

Considering median: we can have k medians from k samples; Let k be 1000;

Sort these k medians and consider central 95% of the medians;

95% CI for population median is ($median_{25}$, $median_{975}$)

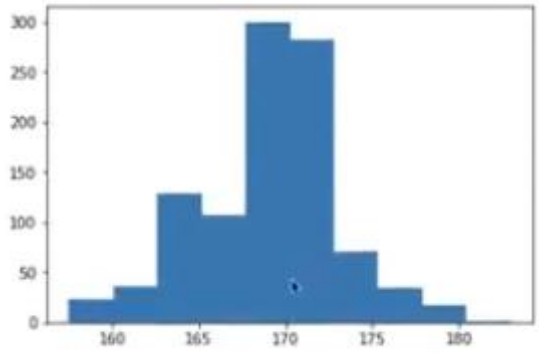
We can use the same process for computing other statistics;

This is a non parametric technique to estimate CI for a statistic; and this process is called as bootstrap;

```
# load dataset
x = numpy.array([180,162,158,172,168,150,171,183,165,176])

# configure bootstrap
n_iterations = 1000
n_size = int(len(x))

# run bootstrap
medians = list()
for i in range(n_iterations):
    # prepare train and test sets
    s = resample(x, n_samples=n_size);
    m = numpy.median(s);
    #print(m)
    medians.append(m)
```

```
# confidence intervals
alpha = 0.95
p = ((1.0-alpha)/2.0) * 100
lower =  numpy.percentile(medians, p)

p = (alpha+((1.0-alpha)/2.0)) * 100
upper =  numpy.percentile(medians, p)
print('%.1f confidence interval %.1f and %.1f' % (alpha*100, lower, upper))
```

2.5




```
95.0 confidence interval 161.5 and 176.0
```

12.28 Hypothesis testing methodology, Null-hypothesis, p-value

Let us have 2 classes; and let the number of students be 50; let us have height values;

Q. Is there a difference in heights of students in cl1 and cl2?

Say we plot histograms and saw that mean of class 2 is greater than mean of class 1;

Hypothesis testing:

1. Choosing a test statistic: ($\mu 2 - \mu 1$)

   If $\mu 2 - \mu 1 = 0$ then the classes have same means;

2. Define a Null hypothesis (H_0): there is no difference in $\mu 2$ and $\mu 1$

   Alternative hypothesis (H_1): there exists a difference in $\mu 2$ and $\mu 1$

3. Compute p-value: probability of observing $\mu 2 - \mu 1$ (that is evident from the observations) if null hypothesis is true;

   Say we have p value = 0.9 for probability of observing a mean difference of 10cm when H_0 is true;

As p value is high than a threshold level (generally 5%) then we accept the null hypothesis else we reject null hypothesis; Also **this p value *does not* tell about the acceptance of alternate hypothesis or the probability of acceptance of null hypothesis;**

Taken an observation of $\mu 2 - \mu 1$ from the classes which is equal to 10 that is $\mu 2 - \mu 1 = 10$ the ground truth statistic that has already occurred; and computing the p value as the conditional probability of finding $\mu 2 - \mu 1 = 10$ given H_0 (which says that there is no difference in means); if the p value is less than threshold that is 5% we reject null hypothesis saying that we cannot accept that there is no difference in means and we reject null hypothesis in favour of the alternate hypothesis; if the p value is > 5% we say that we don't have sufficient the null hypothesis saying that there is sufficient evidence that the observed value occurs under the null hypothesis with satisfactory probability;

P value says that what is the probability of an observation statistic occurs under null hypothesis;

12.29 Hypothesis testing intuition with coin toss example

Example: Given a coin determine if the coin is biased towards heads or not;

Biased towards head: P(H)> 0.5

Not biased: P(H) = 5;

Design an experiment: flip a coin 5 times and count # of heads = X random variable;

This X is the test statistic; (the number of heads)

Perform experiment: Let that we got 5 heads; test statistic X = 5 the observed value;

Let the null hypothesis be "coin is not biased towards head"

$P(X=5 \mid \text{coin is not biased towards head}) = ?$

Probability of observed value given null hypothesis;

$P(X=5 \mid H_0) = (½)^5 \sim 0.03$

$P(X=5 \mid H_0) = 0.03 < 0.05$

Recap: p –value = Probability of observing an observed value given a null hypothesis

If p-value < 0.05 then null hypothesis is incorrect as observation is ground truth it cannot be incorrect; we then reject null hypothesis in favor of alternative hypothesis;

We reject that the coin is unbiased and accept that the coin is biased;

We have sample size as a choice; we have 5 coin flips, we can have 3 flips, 10 flips or 100 flips; and can perform the experiment;

If 3 flips are used and let us observe 3 Heads;

p-value = ½*½*½ = 1/8 = 12.5% > 5%

Thus we fail to reject the null hypothesis; we accept null hypothesis;

Important criterion for hypothesis testing:

1. Design of the experiment
2. Defining Null Hypothesis
3. Design Test statistic

Error: p-value <5% → reject H_0

P-value is the probability of observation given H_0; we cannot say anything about probability of H_0 being true;

12.30 Re-sampling and permutation test

We have the same student heights;

We have 2 classes with 50 height observations each; Take means and compute the difference between the means, let this difference be D;

We then mix all height values into a 100 value vector and randomly sample 50 points from the 100 points to form X vector and rest 50 into Y vector;

We have mean of X and mean of Y: the difference of these means is the mean difference of Sample 1;

Similarly we will generate n samples and compute mean difference; we will have n mean differences; (Note to make random sampling for picking 50 points)

Let n = 10 000;

The mean differences are sorted; We will now have D1, D2, …., D10000 in sorted order;

Now place the original mean difference before sampling in the sorted means list;

Computing the percentage of Di values that are greater D will generate a p-value;

Say D ~ D9500; then we have 500 D values (from D9501 to D10000) that are greater than D; thus we have 5% of values; hence p-value = 5%; we can compute p-value with this process;

Why is this percentage considered as p-value; Initially while jumbling we assumed that there is no difference in the means of the 2 classes or 2 height lists; this is the assumption of null hypothesis; then we have experimented with sampling for 10000 iterations;

If the percentage of the sorted sample differences that are greater than original mean difference D is less than threshold value 5% then the sampling is not random; this implies that null hypothesis should be rejected;

p-value thresholds depends on problem; For medical purposes p-value of 1% is reasonable; generally p-value threshold is kept at 5%;

12.31 K-S Test for similarity of two distributions

Kolmogorov-Smirnov test:

Do two random variables X1 and X2 follow same distribution?

We plot CDF for both random variables;

We will be using hypothesis testing;

Null hypothesis: the two random variables come from same distribution;

Alternative hypothesis: both don't come from same distribution;

Test statistic; D = CDF(X1) – CDF(X2) throughout the CDF range;

= supremum (CDF(X1) – CDF(X2))

= max of (CDF(X1 distribution) – CDF(X2 distribution))

(at same value on x axis)

Null hypothesis is rejected at level α, when D > c(α) * sqrt((n+m)/nm)

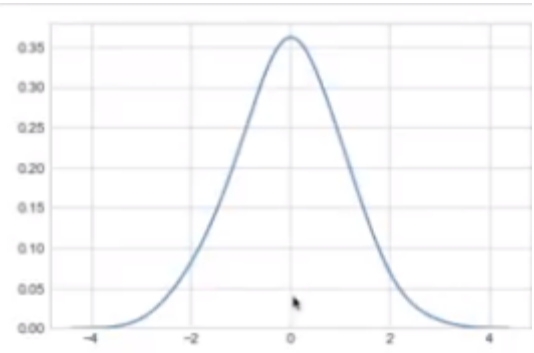
$c(\alpha) = \sqrt{-0.5 * \ln(\alpha/2)}$

$D > (1/n^{0.5}) * (\sqrt{ -0.5 * \ln(\alpha) * (1+ (n/m)) })$

12.32 Code Snippet K-S Test

Normal distribution:

```
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt

#generate a gaussian r.v X
x = stats.norm.rvs(size=1000);
sns.set_style('whitegrid')
sns.kdeplot(np.array(x), bw=0.5)
plt.show()
```



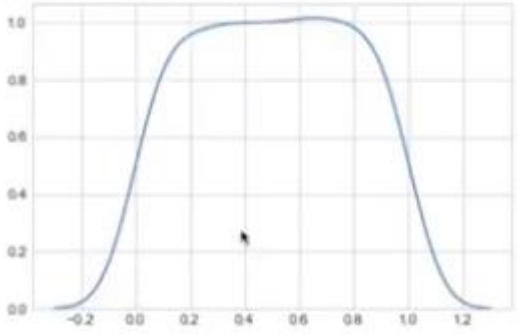

```
stats.kstest(x, 'norm')
```

```
KstestResult(statistic=0.021308397286061931, pvalue=0.75424031453335627)
```

P- value is high thus X comes from Normal distribution;

Uniform distribution:

```
# Y ~ Continous Uniform Distribution(0,1)
y = np.random.uniform(0,1,10000);
sns.kdeplot(np.array(y), bw=0.1)
plt.show()
```




```
stats.kstest(y, 'norm')
```

```
KstestResult(statistic=0.50159644397739489, pvalue=0.0)
```

P-value is low thus X does not follow Normal distribution (we already know that X follows Uniform distribution)

12.33 Hypothesis testing: another example

Difference of means:

Given heights list of two cities, determine if the population means of heights in these two cities is same or not;

Experiment: We cannot collect the whole population data; we will take a sample; say we collect height of 50 random people from both the cities;

Compute sample means from both cities μA (let = 162 cm) and μB (let = 167 cm); these are sample means as population data collection is infeasible;

Test statistic: Mean of city A – Mean of city B = 167 – 62 = 5 cm

Null hypothesis: μB – μA = 0; there is no difference in means of height values of two cities;

Alternative hypothesis: μB – μA != 0

Compute: $P(\mu B - \mu A = 5 \text{ cm} \mid H_0)$ = probability of observing a difference of 5 cm in sample mean heights of sample size 50 between two cities if there is no difference in mean heights;

Case 1: $P(\mu B - \mu A = 5 \text{ cm} \mid H_0) = 0.2 = 20\%$

> There is a 20% chance of observing 5% difference in sample mean heights of C1 and C2 with 50 sample size if there is no population mean difference in heights;
>
> As 20% is significant then null hypothesis must be true; We accept the null hypothesis;

Case 2: $P(\mu B - \mu A = 5 \text{ cm} \mid H_0) = 0.03 = 3\%$

> There is a 3% chance of observing 5% difference in sample mean heights of C1 and C2 with 50 sample size if there is no population mean difference in heights;
>
> As 3% is insignificant then null hypothesis must be False; we reject null hypothesis and accept the alternative hypothesis

12.34 Re-sampling and Permutation test: another example

We have:

> Test statistic: μB – μA = 5 cm with sample size of 50;

Null hypothesis: no difference in population means

List of C1 and C2 heights;

Step 1: S = {C1 U C2} = set of 100 heights

Step 2: From set S, randomly select 50 heights for S1 and 50 heights for S2

Compute means of these S1 and S2 heights: μ1 and μ2; we have re-sampled the data set; with this we are making an assumption that there is no difference in μB and μA, which is the null hypothesis;

Compute: μ2 - μ1

Step 3: Repeat Step 2 for k times, let k = 1000

Result: we will have 1000 (μ2 - μ1); sort these; these are simulated differences under null hypothesis

Result: we will have 1000 (μ2 - μ1) (sorted) and we have μB – μA = 5 cm

Compute the percentage of simulated means that are greater than test statistic; this is the p-value which can be checked for significance at α level;

Note: simulated differences are computed using Null Hypothesis;

P(observed difference$|H_0$) = percentage of simulated means that are greater than test statistic = p-value; if p-value > 5% accept null hypothesis else reject null hypothesis;

Observed difference can never be incorrect as it is ground truth generated from the data; Acceptance or rejection happens with null hypothesis;

12.35 How to use hypothesis testing?

Determining two random variables follow same distribution;

Drug testing: Effectiveness of a new drug over an old drug; claim new drug makes recovery fever from fever in comparison to old drug;

Collect 100 patients: randomly split into two groups of 50 people each;

Administer old drug to group A and new drug to group B; record time taken for all the 100 people to recover;

Let mean time for old drug people be 4 hours and for new drug people be 2 hrs

Mean tells that the new drug is performing well; note that the sample size is 50; thus hypothesis is applied;

H_0: Old drug and new drug take same time for recovery;

Test statistic: $\mu_{old} - \mu_{new} = 2$;

$P(\mu_{old} - \mu_{new} = 2 \mid H_0) = ?$ (let = 1%)

If there is no difference in old drug and new drug then the probability of observing $\mu_{old} - \mu_{new} = 2$is 1%; this implies that the null hypothesis and observation do not agree with each other, thus null hypothesis is incorrect;

P-value: agreement between null hypothesis and the test statistic;

Probability value is computed using re-sampling and permutation test; combine the datasets into 1 large set and sample randomly and compute test statistics for samples and sort the results; find the percentage of values that are greater than the original test statistic value;

Significance level is problem specific; for problems such as medical tests significance level is 0.1% or1%; for ecommerce 3% or 5% are generally taken;

Can be used for computing average spending of a set of people;

12.36 Proportional Sampling

Let d: [d1, d2, d3, d4, d5] = [2.0, 6.0, 1.2, 5.8, 20.0]

Task: pick an element from the list such that the probability of picking the element is proportional to the value;

For random selection the probability of picking any value is equal to all other values;

But the task requires proportional sampling:

1. a. Compute sum of all the elements in the list: = 35.0

   b. Divide list with the sum, list = [0.0571, 0.171428, 0.0343, 0.1657, 0.5714]

      Sum of all these values = 1 and all are between 0 and 1

   c. Cumulate the list:

      C_list = [0.0571, 0.228528, 0.262828, 0.428528, 1.00]

2. Sample one value from Uniform distribution between 0 and 1; let r = 0.6;
3. Use C_list; place r in the list and return the original value at the index at which r is placed;

In C_list the gap between values is proportional to the original values; as random number r is generated from a uniform distribution; the probability of picking any value is equal to the gap between the elements in C_list in turn proportional to the original list values;

12.37 Revision Questions

1. What is PDF?
2. What is CDF?
3. Explain about 1-std-dev, 2-std-dev, 3-std-dev range?
4. What is Symmetric distribution, Skewness and Kurtosis?
5. How to do Standard normal variate (z) and standardization?
6. What is Kernel density estimation?
7. Importance of Sampling distribution & Central Limit theorem
8. Importance of Q-Q Plot: Is a given random variable Gaussian distributed?
9. What is Uniform Distribution and random number generators
10. What Discrete and Continuous Uniform distributions?
11. How to randomly sample data points?
12. Explain about Bernoulli and Binomial distribution?
13. What is Log-normal and power law distribution?
14. What is Power-law & Pareto distributions: PDF, examples
15. Explain about Box-Cox/Power transform?
16. What is Co-variance?
17. Importance of Pearson Correlation Coefficient?
18. Importance Spearman Rank Correlation Coefficient?
19. Correlation vs Causation?
20. What is Confidence Intervals?
21. Confidence Interval vs Point estimate?
22. Explain about Hypothesis testing?
23. Define Hypothesis Testing methodology, Null-hypothesis, test-statistic, p-value?
24. How to do K-S Test for similarity of two distributions?

Chapter 13: Interview Questions on Probability and statistics

13.1 Questions and Answers

1. What is a random variable?
2. What are the conditions for a function to be a probability mass function?(http://www.statisticshowto.com/probability-mass-function-pmf/)
3. What are the conditions for a function to be a probability density function ?(Covered in our videos)
4. What is conditional probability?
5. State the Chain rule of conditional probabilities?(https://en.wikipedia.org/wiki/Chain_rule_(probability))
6. What are the conditions for independence and conditional independence of two random variables?(https://math.stackexchange.com/questions/22407/independence-and-conditional-independence-between-random-variables)
7. What are expectation, variance and covariance?(Covered in our videos)
8. Compare covariance and independence?(https://stats.stackexchange.com/questions/12842/covariance-and-independence)
9. What is the covariance for a vector of random variables?(https://math.stackexchange.com/questions/2697376/find-the-covariance-matrix-of-a-vector-of-random-variables)
10. What is a Bernoulli distribution?
11. What is a normal distribution?
12. What is the central limit theorem?
13. Write the formula for Bayes rule?
14. If two random variables are related in a deterministic way, how are the PDFs related?
15. What is Kullback-Leibler (KL) divergence?
16. Can KL divergence be used as a distance measure?
17. What is Bayes' Theorem? How is it useful in a machine learning context?
18. Why is "Naive" Bayes naive?
19. What's a Fourier transform?
20. What is the difference between covariance and correlation?
21. Is it possible capture the correlation between continuous and categorical variable? If yes, how?
22. What is the Box-Cox transformation used for?
23. What does P-value signify about the statistical data?
24. A test has a true positive rate of 100% and false positive rate of 5%. There is a population with a 1/1000 rate of having the condition the test identifies. Considering a positive test, what is the probability of having that condition?
25. How you can make data normal using Box-Cox transformation?
26. Explain about the box cox transformation in regression models.
27. What is the difference between skewed and uniform distribution?
28. What do you understand by Hypothesis in the content of Machine Learning?

29. How will you find the correlation between a categorical variable and a continuous variable?
30. How to sample from a Normal Distribution with known mean and variance?

Chapter 14: Dimensionality Reduction and Visualization

14.1 What is Dimensionality Reduction?

We can visualize 2D and 3D using Scatter Plots; up to 6D we can leverage Pair Plots. For nD > 6D we should reduce the dimensionality to make it an understandable or a visualizable dataset.

Techniques: tSNE and PCA

By reducing dimensionality, we transform the features into a new set of features which are less in number. The aim lies in preserving the variance which ensures least possible loss of information.

14.2 Row Vector and Column Vector

X_i belongs to R(d) → X_i is a d dimensional column vector and several data points are stacked in row wise to form a dataset.

14.3 How to represent a data set?

D = {x_i, y_i}, x_i belongs to R(d), y_i = R(1)

x_i: data points, y_i: class labels

14.4 How to represent a dataset as a Matrix?

Matrix is like a table

D = {x_i, y_i}, i = [1 to n], x_i belongs to R(d), y_i class labels

x_i is a column vector

In a matrix: each row is a data point and each column vector is a feature (preferred form of representation, this is one type of representation scheme)

Y is a column matrix in which each row corresponds to the class that the data point x_i belongs to.

14.5 Data Preprocessing: Feature Normalization

Data preprocessing: Mathematical transformations on Data to make it Machine readable improving its quality and reducing the dimensionality (sometimes).

Column Normalization: Take each column, corresponding to a feature. Take all values and do minimum subtraction and range division for each feature separately.

- a_max, a_min: Calculated or generated
- From a1, a2, a3, …, an make transformations to a1', a2', a3', ….., an' such that

  ai' = (ai – a_min)/(a_max – a_min)
- The feature normalization method transforms all the features to a range from 0 to 1.
- Column Normalization is needed to avoid different feature scales, Some of the variables can take values between 1 Million and 2 Million, while some variables or features can only take values between 0 and 10. Thus, the second feature becomes insignificant to a model as larger values will effect largely during the predictions stage of the model.
- Through column normalization we get rid of feature scales. This squishes all the data into a unit hyper cube.

14.6 Mean of a data matrix

x_mean = (1/n) * sum (I = 1 to n) x_i

Geometrically: Project all data points onto each feature axis. The central vector of all these projections is the mean on each feature axis. The set of the means is the mean vector of the data matrix.

14.7 Data Pre-processing: Column Standardization

Column Normalization: values are squished to a unit hyper-cube [0, 1], this gets rid of scales of each feature

Column Standardization: In a data matrix, with rows as data points and columns as features

a1, a2, a3, a4, …., an for each feature: are transformed to a1', a2', a3', ….., an' such that

ai' = (ai – a_mean) / a_std

With standardization, we will have mean =0 and std_dev = 1

Column Standardization = Mean Centering + unit scaling

Geometrically: Project all data points on feature axes; we can get mean and standard deviation of the dataset. After applying column standardization the data points will have mean at origin and the data points are transformed to a unit standard deviation.

Why standard deviation is important?

14.8 Co-variance of a Data Matrix

X = n x d matrix, its co-variance matrix S is of d x d size

f_j = column vector – jth feature

S_{ij} = ith row and jth column in co-variance matrix, is a square matrix

x_{ij} = value of ith datapoint and jth feature

$S_{ij} = cov(f_i, f_j)$

i: 1→ d, j: 1→ d

$cov(x,y) = (1/n) * sum(1 \text{ to } n) (x_i – mean\_x)*(y_i – mean\_y)$

$cov(x,x) = var(x)$ -- 1

$cov(f_i, f_j) = cov(f_i, f_i)$ -- 2

In co-variance matrix, we have diagonal elements as variance values and the matrix is symmetric.

Let X: column standardized, $mean(f_i) = 0$, $std\_dev(f_i) = 1$

$Cov(f_1, f_2) = (1/n)*sum(1 \text{ to } n) (x_{i1} * x_{i2}) = (f_1 . f_2)/n$

$S(X) = (X^T X) *1 /n$: if X has been column standardized

14.9 MNIST dataset (784 dimensional)

D = { x_i, y_i } i = 1 to 60k

Each data point: x_i = Image (28 x28), y_i = {0, 1…., 9}

Dimensionality reduction: from 2D to 1D using Flattening operation

(28 x 28) is converted into (784 x 1) shape; by stacking all elements from each row into a column vector

We can use PCA or tSNE to visualize the dataset using a plot

14.10 Code to load MNIST data set

```
import numpy as np

import pandas as pd
```

```
import matplotlib.pyplot as plt

d0 = pd.read_csv('') #csv – comma separated values

print(d0.head())

l = d0['label']

d = d0.drop("label", axis = 1)


plt.figure(figuresize = (7, 7))

idx = 100

grid_data = d.iloc(idx).as_matrix.reshape(28, 28)

plt.imshow(grid_data, interpolation = "name", cmap = "gray")

plt.show()

print(l(idx))
```

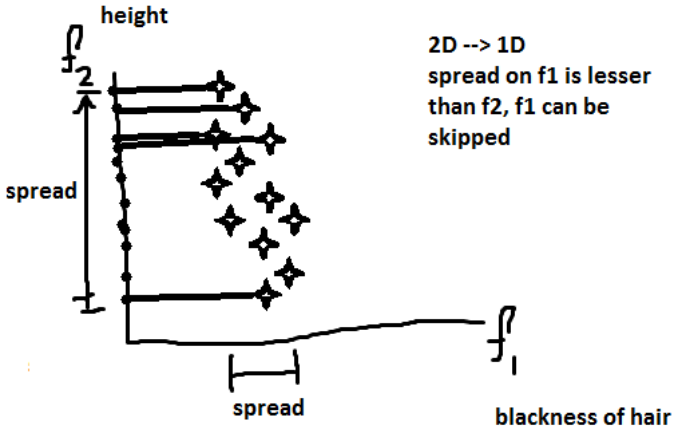
Chapter 15: PCA (principal component analysis)

15.1 Why learn PCA?

PCA – for dimensionality reduction - R(d) to R(d') where d' << d

1) For visualization
2) d' < d for model training

15.2 Geometric intuition of PCA

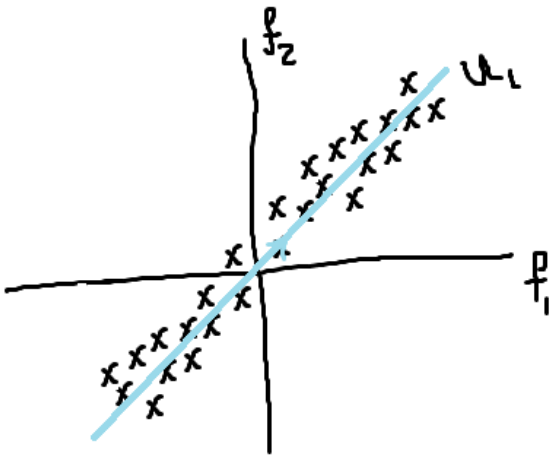


If we want to reduce dimensionality from 2D to 1D, we can skip feature f_1 as the spread across the feature is very less than spread across the feature f_2.

Another case: Let y = x be the underlying relationship between two features and let X be column standardized, and the plot is such that there is sufficient variance on both features. Say, we rotate the axes and reach an orientation where spread on f_2' << f_1' (perpendicular to f2') then we can drop f_2'. So data transformation can also find maximum variance features. This will help us reduce dimensionality.

We want to find a direction f_1' such that the variance of the data points projected onto f_1' is maximum and skip features with minimum spread.

15.3 Mathematical objective function of PCA



u_1 : Unit vector: in the direction where maximum variance of projection of x_i exists

$|| u_1||^2 = 1$

x_i' $= proj_{u1}\ x_i$ (projection of x_i on u_1)

$= u_1 . x_i / || u_1||^2 = u_1^T x_i$

x_i' $= u_1^T x_i$

x_i' (mean) $= u_1^T x_{mean}$

Task: find u_1, such that $var(proj_{u1}\ x_i)$ is maximum

$var\{u_1^T x_i\}$ (i = 1 to n) = (1/n) * sum $(u_1^T x_i - u_1^T x_{mean})^2$

When column standardized:

$var\{x_i' = u_1^T x_i\}$ (i = 1 to n) = (1/n) * sum $(u_1^T x_i)^2$

Optimization: Objective: max (1/n)*sum $(u_1^T x_i)^2$ such that

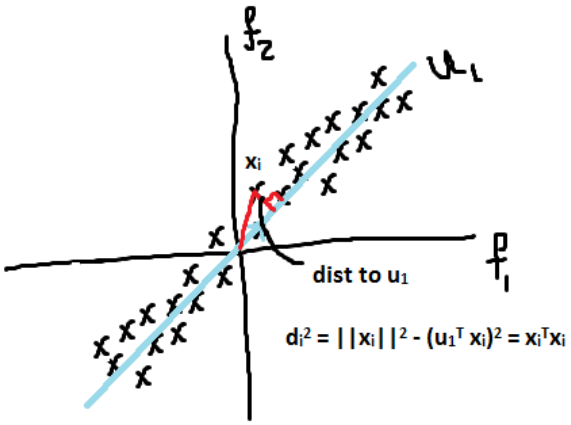
Constraint: $u_1^T u_1 = 1 = || u_1||^2$

15.4 Alternative formulation of PCA: Distance Normalization

Variance minimization: find u_1 such that projected variance is maximum

Alternatively, find u_1 such that distances of the data points to u_1 is minimum

min sum(d_i^2) for u_1: u_1 = unit vector



Variance maximization:

Objective: max (1/n)*sum $(u_1^T x_i)^2$ such that

Constraint: $u_1^T u_1 = 1 = ||u_1||^2$

Distance minimization:

Objective: min $(||x_i||^2 - (u_1^T x_i)^2)$ = min sum$(x_i^T x_i - (u_1^T x_i)^2)$

Constraint: $u_1^T u_1 = 1 = ||u_1||^2$

Same u_1 optimizes both formulations

15.5 Eigen values and Eigen vectors (PCA): Dimensionality reduction

Solution to the optimization problems: λ_1, V_1

X: [……]$_{n \times d}$: column standardized: mean of each column = 0 and std_dev of each column = 1

Co-variance matrix of X = S

$S_{dxd} = X_{dxn}^T X_{nxd}$: S is square and symmetric

Eigen value of S: $\lambda_1, \lambda_2, \lambda_3, \lambda_4$, …. ($\lambda_1 > \lambda_2 > \lambda_3 > \lambda_4$ ….)

Eigen vectors of S: V_1, V_2, V_3, V_4, …

Def: If $\lambda_1 V_1 = S_{dxd} V_1$

λ_1 is a scalar which is the Eigen value of S and V1 is the Eigen vector corresponding to λ_1

Eigen vectors are perpendicular to each other. This implies, that dot product of pairs of Eigen vectors is zero. $V_i^T V_j = 0$

$u_1 = V_1$ = Eigen vector of S (= $X^T X$) corresponding to largest Eigen vector (=λ_1)

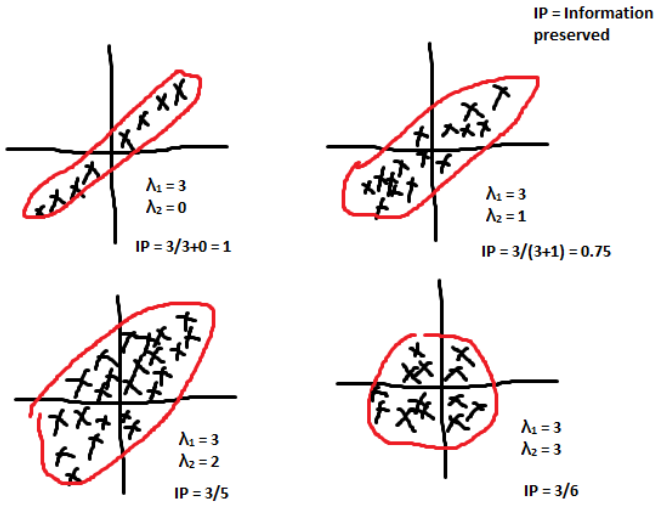
Steps:

1. X: Column Standardized
2. $S = X^T X$
3. λ_1, V_1 = eigen(s)
4. $u_1 = V_1$ direction with maximum variance

λ_1/ sum(λ_i) = % of variance preserved on u_1

V_1 max variance, V_2 second max var, V_3 third max var, …

Eigen Vectors provide direction of maximum variance, Eigen values provides percentage of variance preserved with each Eigen value



15.6 PCA for Dimensionality Reduction and Visualization

For X: Let us have S, V_1, and V_2; using eigen(s) function

Transform $x_i = [f_{1i}, f_{2i}]$ to $x_i' = [x_i^T V_1, x_i^T V_2]$, we can drop $x_i^T V_2$ as there is not much variance: 2D data point is transformed to 1D

If we want to preserve say k% of variance:

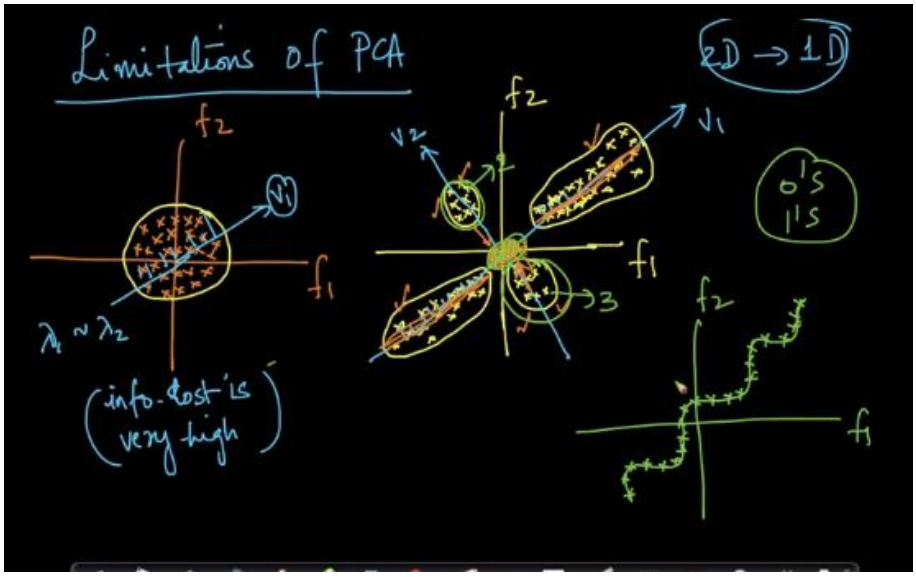
Find d' such that sum(i = 1 to d') λ_i / sum(i = 1 to d) λ_i = k% where d is dimensionality of the input data point.

15.7 Visualize MNIST dataset

https://colah.github.io/posts/2014-10-Visualizing-MNIST/

15.8 Limitations of PCA

Loss of information: cases for hyper-spheres, separable data points on high dimensionality can get crowded at several locations making it inseparable at low dimensionality



15.9 PCA code example

MNIST dataset:

Step 1: Data preprocessing

- Standardization: using StandardScaler() = X, mean centering and scaling

- Get $(1/n)*X^TX$ – Co-variance matrix

- values, vectors = eigh(covar_matrix, eigvals = (782, 783)) # gives top 2 values as co-variance matrix is of 784 x 784 size

- vectors = $vectors^T$

- X' = vectors*X

Using PCA:

- from sklearn import decomposition
- pca = decomposition.PCA()
- pca.n_components = 2
- pca_data = pca.fit_transform(sample_data)
- pca_data = np.vstack(pca_data.T, labels)).T

15.10 PCA for dimensionality reduction (not-visualization)

d' = 2 or 3 is for visualization, but if d'= 200 or any other value, then we are doing data reduction for non-visualization tasks.

Stack Eigen vectors horizontally to get Eigen matrix V. Multiply X with V to get X' which is dimensionally reduced. d' can be determined by checking explained variance. The task is to maximize variance.

Explained variance = % of variance retained = $100*\text{sum}(i = 1 \text{ to } d') \lambda_i / \text{sum}(i = 1 \text{ to } d) \lambda_i$

Take pca.n_components = 784

pca_data = pca.fit_transform(sample_data)

percentage_var_explained = np.cumsum(percentage_var_explained)

Chapter 16: (t-SNE) T-distributed Stochastic Neighborhood Embedding

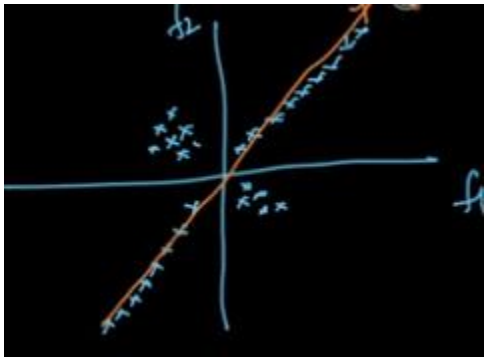
16.1 What is t-SNE?

One of the best techniques for data visualization: tSNE

PCA – basic, old: did not perform well on MNIST dataset visualization

Other techniques: Multiple Dimensional Scaling, Sammon Mapping, Graph based techniques

tSNE vs PCA:



PCA preserves the global structure of the dataset and discards local structure. tSNE also preserves local structure of the dataset.

16.2 Neighborhood of a point, Embedding

For a d-dimension space:

> Neighborhood points of a data point are the points that are geometrically close to the data point.
>
> Embedding: For every point in the original dataset, a new point is created in lower dimension corresponding to the original data point. Embedding means projecting each and every input data point into another more convenient representation space (Picking a point in high dimensional space and placing it in a low dimensional space).

16.3 Geometric intuition of t-SNE

Let us assume: x1, x2, x3, x4 and x5 are some data points, and let x1, x2 and x3 be in a neighborhood region, x4 and x5 be another neighborhood region and both

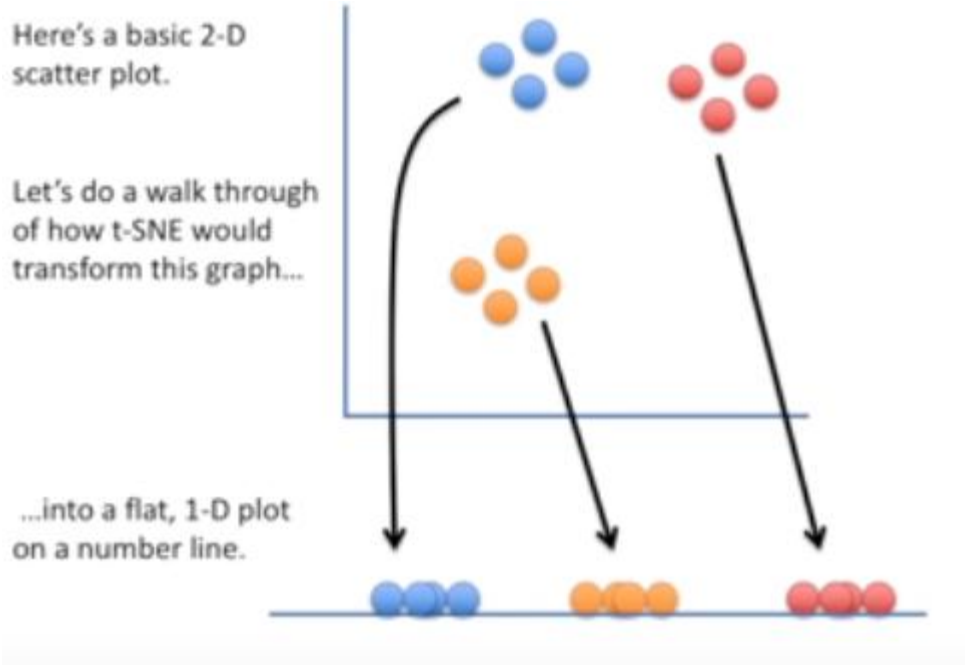
neighborhood regions are very far from each other. When we reduce the dimensionality we follow the objective of preserving the distance between data points.

dist(x1, x2) ~ dist(x1', x2')

For points which are not in neighborhood the distances are not preserved.

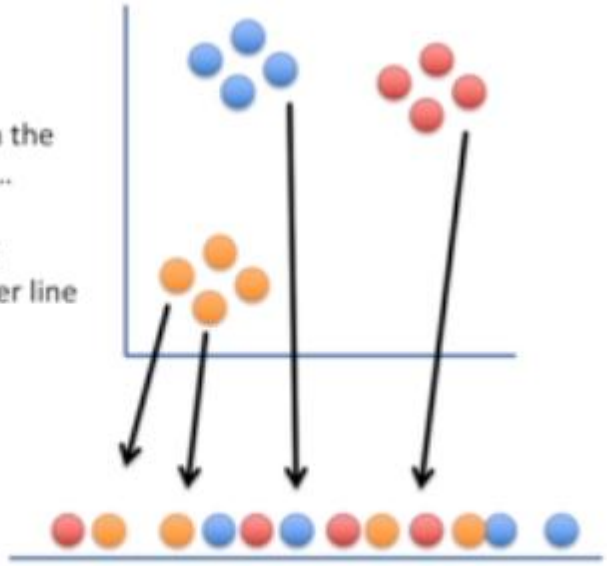
Relation between dist(x1, x4) and dist(x1', x4') is ignored.

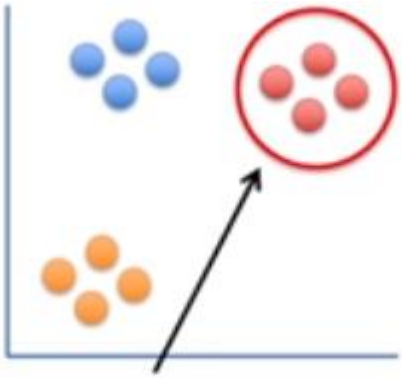
Link: https://www.youtube.com/watch?v=NEaUSP4YerM



We'll start start with the original scatter plot...

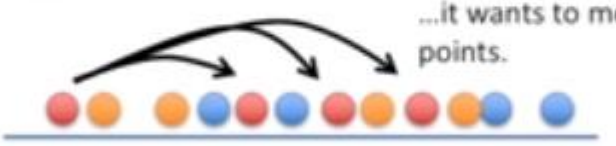
... then we'll put the points on the number line in a random order.

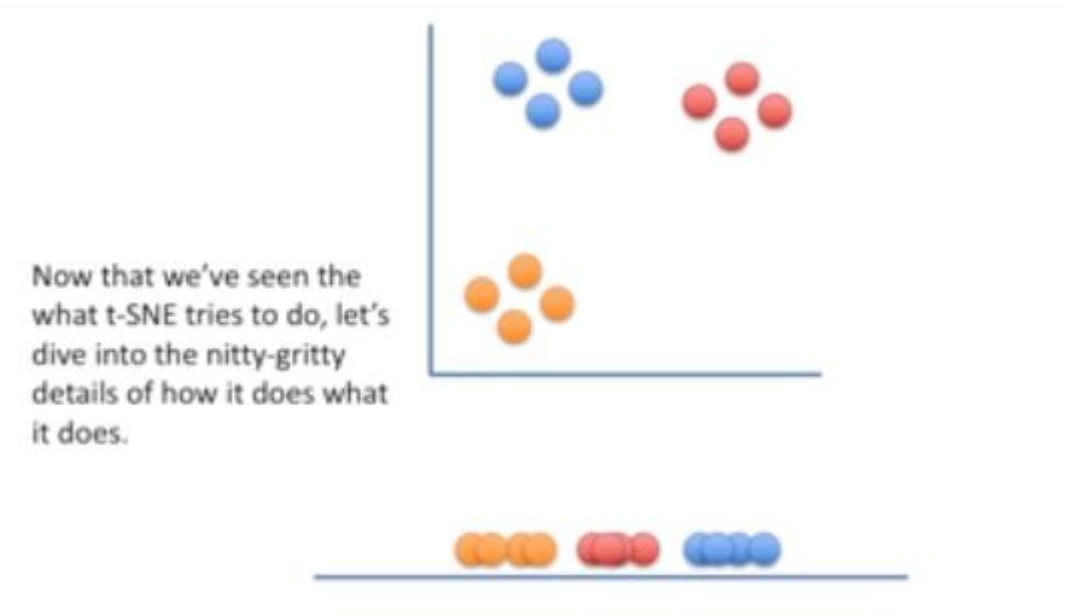




Because it is part of this cluster...

...it wants to move closer to these points.





Link: https://www.youtube.com/watch?v=ohQXphVSEQM

Let x_i be the i^{th} object in high dimensional space
Let y_i be the i^{th} object in low dimensional space
Construct:

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)}$$

$$q_{j|i} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq i} exp(-||y_i - y_k||^2)}$$

$p_{j|i}$ = conditional probability of two points being in neighborhood in high dimension,

$q_{j|i}$ is for low dimension

Match these functions by minimizing sum of Kullback-Leibler divergences:

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log\left(\frac{p_{j|i}}{q_{j|i}}\right)$$

KL divergence is asymmetric: gives large cost for representing nearby data points in high dimensional space by widely separated points in low dimensional space.

Perplexity indicates number of neighbors considered during analysis.

$$\frac{\delta C}{\delta y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j)$$

For optimization, we can use Gradient Descent.

16.4 Crowding Problem

Non-neighbor data points can also be seen to crowd with neighborhood data points.

t-SNE preserves distances in a Neighborhood. It is impossible to preserve neighborhood of all the data points.

16.5 How to apply t-SNE and interpret its output?

Link: https://distill.pub/2016/misread-tsne/

Processes all the data in multiple iterations and tries to separate different neighborhoods.

Parameters:

Step: Number of iterations

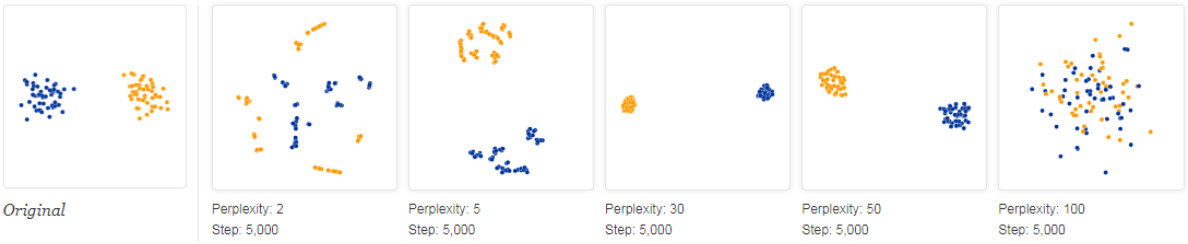
Perplexity: Loosely indicates number of neighbors to whom distances are preserved.

Perplexity: Run tSNE with multiple perplexity values. As perplexity increases from 1, the neighborhood tries to get good clusters and then with further increments the clustered profile becomes a mess (more number of data points are considered to belong to same neighborhood).

Stochastic part of tSNE induces probabilistic embedding, results change every time tSNE is run on the same data points with same parameter values.

tSNE also expands dense clusters and shrinks sparse clusters, cluster sizes cannot be compared. tSNE does not preserve distances between clusters.

Never rely on the results of tSNE, experiment it with changing parameter values. You may need to plot multiple plots to visualize.



16.6 t-SNE on MNIST

MNIST: 784-dim data of images of hand written Numbers between 0 and 9.

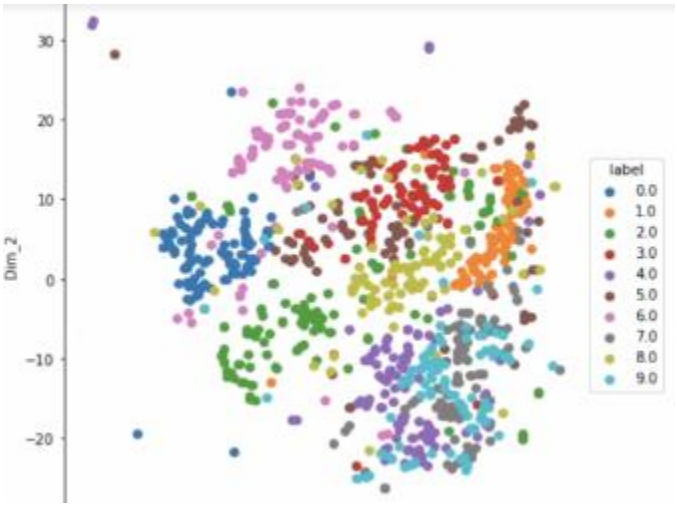
tSNE grouped data points or images based on visual similarity;

16.7 Code example of t-SNE

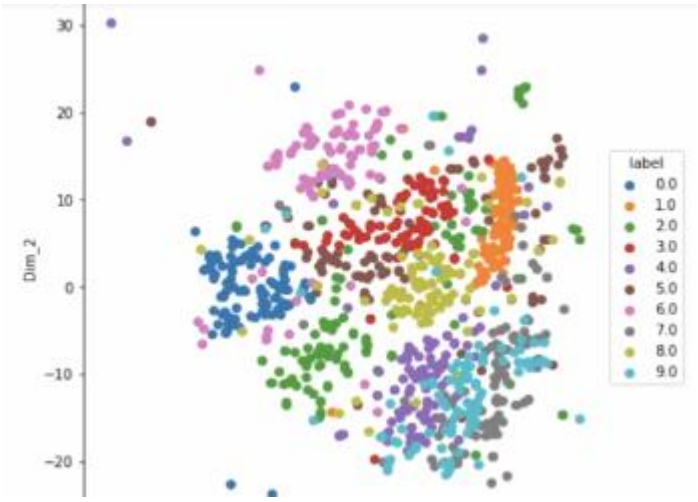
Model = TSNE(n_components = 2, random_state = 0)

Tsne_data = Model.fit_transform(data_1000)

Plot: perplexity = 30, iter = 1000



Perplexity = 50, iter = 5000, data points = 1000



16.8 Revision Questions

1. What is dimensionality reduction?
2. Explain Principal Component Analysis?
3. Importance of PCA?
4. Limitations of PCA?
5. What is t-SNE?
6. What is Crowding problem?
7. How to apply t-SNE and interpret its output?

Chapter 17: Interview Questions on Dimensionality Reduction

17.1 Questions & Answers

1. You are given a train data set having 1000 columns and 1 million rows. The data set is based on a classification problem. Your manager has asked you to reduce the dimension of this data so that model computation time can be reduced. Your machine has memory constraints. What would you do? (You are free to make practical assumptions.)(https://www.analyticsvidhya.com/blog/2016/09/40-interview-questions-asked-at-startups-in-machine-learning-data-science/)

Ans. Close all other applications, random sample dataset, reduce dimensionality (remove correlated features, Use correlation for Numerical and chi-square test for categorical, Can use PCA)

2. Is rotation necessary in PCA? If yes, Why? https://google-interview-hacks.blogspot.com/2017/04/is-rotation-necessary-in-pca-if-yes-why.html

Ans. Yes, rotation (orthogonal) is necessary because it maximizes the difference between variance captured by the component. This makes the components easier to interpret. Not to forget, that's the motive of doing PCA where, we aim to select fewer components (than features) which can explain the maximum variance in the data set. By doing rotation, the relative location of the components doesn't change; it only changes the actual coordinates of the points.

If we don't rotate the components, the effect of PCA will diminish and we'll have to select more number of components to explain variance in the data set.

3. You are given a data set. The data set contains many variables, some of which are highly correlated and you know about it. Your manager has asked you to run PCA. Would you remove correlated variables first? Why?(https://www.linkedin.com/pulse/questions-machine-learning-statistics-can-you-answer-saraswat/)

Ans. Since, the data is spread across median, let's assume it's a normal distribution. We know, in a normal distribution, ~68% of the data lies in 1 standard deviation from mean (or mode, median), which leaves ~32% of the data unaffected. Therefore, ~32% of the data would remain unaffected by missing values.

Chapter 18: REAL WORLD PROBLEM: PREDICT RATING GIVEN PRODUCT REVIEWS ON AMAZON

18.1 DATASET OVERVIEW: AMAZON FINE FOOD REVIEWS (EDA)

Amazon largest retailers of the world;

The dataset consists of reviews of fine foods on Amazon;

Half a million reviews over 13 years;

Contains: Review ID, ProductId, UserId, ProfileName, Text, Summary, Score, etc.

Task: Given a review determine whether the review is positive; this helps whether we need to do any improvements in the product;

18.2 DATA CLEANING: DEDUPLICATION

Garbage in garbage out;

In fine food reviews, we have:

A single person giving reviews to multiple at the same time stamp and same review; these can be thought of as duplicate data;

pd.DataFrame.drop_duplicates(list of features)

Other such errors in data needs to be found out; such as values that need to be less than 1, data types, etc.

18.3 WHY CONVERT TEXT TO A VECTOR?

Linear algebra can be applied when the input is in numerical data type;

Converting review text strings into a d dimensional numerical vector;

If we plot the data points of n dimensional vector (including vectorized) text data, we can utilize linear algebra in terms of distance measurements and the side to which the data points exist with respect to a plane that separates the positive data points from negative data points;

$w^T x_1 > 0$ then review is positive else negative;

If similarity between r1 and r2 is more than similarity between r1 and r3, then r1 and r2 are closer than r1 and r3 (in terms of distances); similar texts must be closer geometrically;

18.4 BAG OF WORDS (BOW)

Example:

R1: This pasta is very tasty and affordable;

R2: This pasta is not tasty and is affordable;

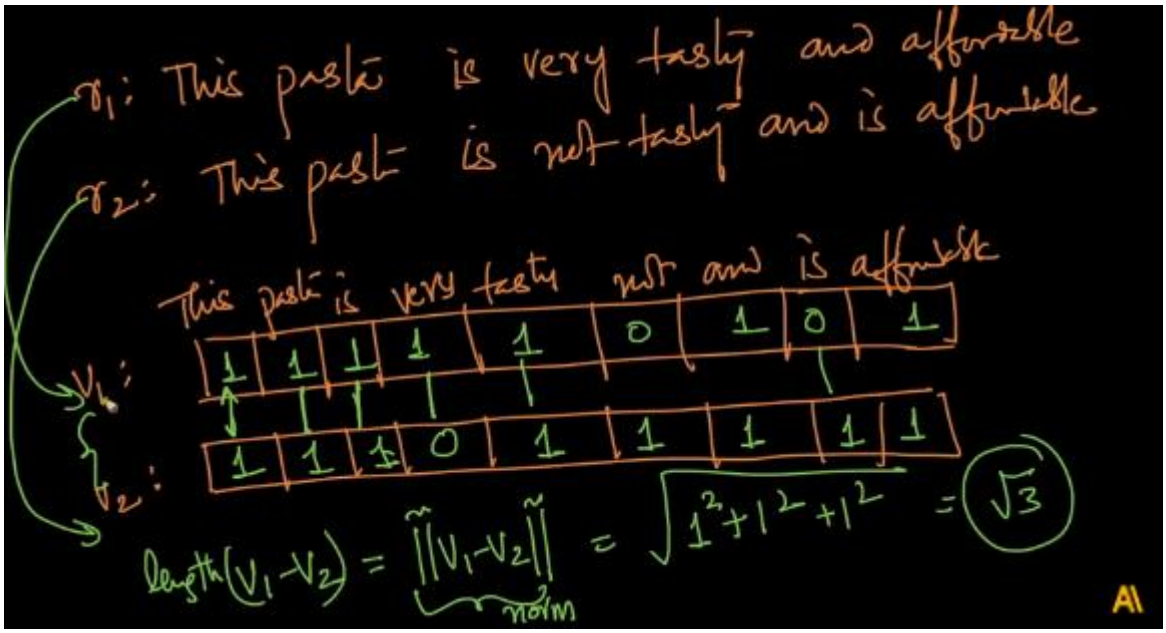
R3: This pasta is delicious and cheap

R4: Pasta is tasty and pasta tastes good;

BOW: Construct a dictionary of all unique words in the reviews;

d – Unique words: for n reviews or n documents;

Construct a vector of size d; each element in the vector belongs to words such as ‘a’, ‘an’, ‘the’, ‘pasta’, …, ‘tasty’, ….



Most of the elements in the vector are zero; thus we will have a sparse matrix;

BOW: The reviews R1 and R2 are completely opposite but the data points are closer to each other; BOW does not perform well when there is a small change in the data;

We can have a binary bag of words;

BOW depend on count of words in each review, this discards the semantic meaning of the documents and documents that are completely opposite in meaning can lie closer to each other when there is very small change in the document (with respect to the words)

18.5 TEXT PREPROCESSING: STEMMING, STOP-WORD REMOVAL, TOKENIZATION, LEMMATIZATION.

R1: This pasta is very tasty and affordable;

R2: This pasta is not tasty and is affordable;

R3: This pasta is delicious and cheap

R4: Pasta is tasty and pasta tastes good;

The underlined words above are stop words which do not add any value to the semantic meaning of the document;

Now with this we will have sparsity reduced; (nltk tool kit has stop words)

We should convert everything in **lowercase**;

Next we can **remove stop words**

Stemming: Tastes, tasty and tasteful: indicate the same meaning which relates to taste;

With stemming we can combine words which have same root words;

Two of the algorithms are: Porter Stemmer and Snowball Stemmer;

Snowball stemmer is more powerful

Tolenization: Breaking up a sentence into words;

Lemmatization takes lemma of a word based on its intended meaning depending on the context of surrounding words;

18.6 UNI-GRAM, BI-GRAM, N-GRAMS.

R1: This pasta is very tasty and affordable;

R2: This pasta is not tasty and is affordable;

After data cleaning:

R1: pasta tasty affordable

R2: pasta tasty affordable

Uni grams consider each single word for counting; Bi grams considers two consecutive words at a time; We can have n grams similarly;

Uni grams based BOW discards sequence of information; with n grams we can retain some of the sequence information;

Dimensionality of the vector increases when including n grams;

18.7 TF-IDF (TERM FREQUENCY - INVERSE DOCUMENT FREQUENCY)

BOW: one way to vectorize text data: using n grams, preprocessing and there were variations of bag of words;

TF: Term frequency;

TF(Wi, rj) = # of times occurs in rj / total number of words in rj; can also be thought of as probability of finding a word Wi in a document rj;

IDF(Wi, Dc) = log(N/ni)

rj: jth review, Dc: Document corpus, Wi: ith word; N: #of documents, ni = # docs which contain Wi;

N/ni >=1; log(N/ni) >= 0 ; IDF >= 0

If ni increases log(N/ni) decreases;

If Wi is more frequent in corpus then its IDF is less;

Rare words in documents that occur more frequently in a review will have high TF-IDF value; TFIDF = tf(Wi, rj) * idf(Wi, Dc);

More importance to rarer words in corpus and give more importance to words that are frequent in a review;

18.8 WHY USE LOG IN IDF?

IDF(Wi, Dc) = log(N/ni);

Usage of log can be understood from Zipf's law;

> Words occurrences in English or any language follows a power law distribution, some words such as 'the' and 'in' are frequently found in usage, while such as geometry and civilization occur less in text documents;

When a log – log plot is applied we can find a straight line;

IDF without a log will have large numbers that will dominate in the ML model;

Taking log will bring down dominance of IDF in TFIDF values;

18.9 WORD2VEC.

This technique takes semantic meaning of sentences;

Word is transformed into a d dimension vector; This is not a sparse vector and generally has dimensionality << BOW and TFIDF

If two words are semantically similar, then vectors of these words are closer geometrically;

Word2vec also retains relationships between words such as King:Queen::Man:Woman

> Vman – Vwoman || Vking – Vqueen (|| - parallel vectors)
>
> Vitaly – Vrome || Vgermany – Vberlin
>
> Vwalking – Vwalked || Vswimming – Vswam
>
> Word2vec learns these properties automatically; Why part of the Word2vec can be learnt in Matrix Factorization;

Working: Takes a text corpus (large size): for every word its builds a vector: dimension of the vectors is given as an input; this uses words in the neighborhood: if the neighborhood is similar then the words have similar vectors;

Google took data corpus from Google News to train its Word2Vec method;

18.10 AVG-WORD2VEC, TF-IDF WEIGHTED WORD2VEC

Each review has a sequence of words/sentences

Vector V1 of review r1: avg w2v(r1) = (1/# words in r1) (w2v(w1) + w2v(w2) + ….)

Tfidf w2v (r1) = (tfidf(w1)*w2v(w1) + tfidf(w2)*w2v(w2) + …) / (tfidf(w1) + tfidf(w2) + …)

18.11 BAG OF WORDS (CODE SAMPLE)

Sklearn's CountVectorizer()

Counts of each word in the corpus;

Sparsity = # of zero elements / total elements

With sparse matrices, we can store sparse matrices with a memory efficient method by storing the row and column indices with corresponding values;

18.12 TEXT PREPROCESSING (CODE SAMPLE)

Removing html tags, punctuations, stop words, and alpha numeric words, ensuring number of letters >2, lowercases, using stemming and lemmatization;

Python module re: regular expressions; Can be used extensively for text preprocessing;

18.13 BI-GRAMS AND N-GRAMS (CODE SAMPLE)

Sklearn's CountVectorizer(ngram_range = (1,2))

18.14 TF-IDF (CODE SAMPLE)

Sklearn's TfidfVectorizer(ngram_range = (1,2))

18.15 WORD2VEC (CODE SAMPLE)

Using gensim module: stemming words may not have w2v vectors in pre trained model

18.16 AVG-WORD2VEC AND TFIDF-WORD2VEC (CODE SAMPLE)

We use formulae to compute these vectors; Achieved using a for loop;

Chapter 19: CLASSIFICATION AND REGRESSION MODELS: K-NEAREST NEIGHBORS

19.1 HOW "CLASSIFICATION" WORKS?

Working of classification:

We are using text which is most informative; text is converted into a vector and based on vector we have either positive or negative reviews;

Task of classification is to find a function that takes input text vector and gives output as positive or negative for any new review text;

We are classifying a new review into two classes positive or negative;

Most of Machine Learning is finding a mathematical function;

$Y = F(X)$

The classification model gets training data and the model learns the function F observing examples; then this learnt function is used to predict the output of new unseen input data; Unseen data implies that the data is not used during training;

19.2 DATA MATRIX NOTATION

Given input matrix X; each row X_i represents a review text vector; for each review we will have a class label which is denoted as Y_i;

A vector is by default a column vector;

In Machine Learning we will leverage Linear Algebra for computations; We cannot input text into Linear Algebra; thus even y value are vectorized; here we have two classes thus we can have a binary label 0 or 1; 1 representing positive review, 0 for negative review;

19.3 CLASSIFICATION VS REGRESSION (EXAMPLES)

$D_n = \{(x_i, y_i) \mid x_i \in R^d, y_i = \{0, 1\}\}$: Binary classification

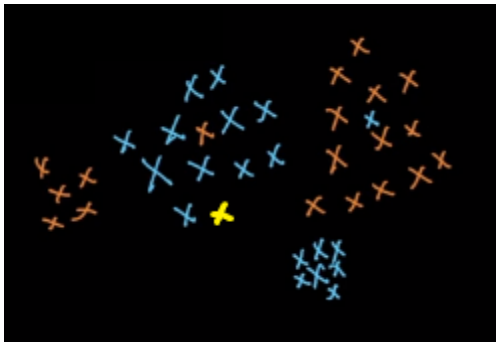
For MNIST: $D_n = \{(x_i, y_i) \mid x_i \in R^{nxn}, y_i = \{0, 1, 2, \ldots\ldots, 9\}\}$: Multi class classification

Regression: Heights of a school of students: a continuous random variable: Regression

19.4 K-NEAREST NEIGHBOURS GEOMETRIC INTUITION WITH A TOY EXAMPLE

Simple and a powerful Machine Learning Algorithm: Classification and Regression

2D toy Dataset:



Blue points: + ve data points; Orange points: - ve data points; Yellow: query point (x_q)

Task: Predict the class of query point: It can be classified into a class based on the classes of the neighbors of the query data point;

1. Find 'k' nearest points to x_q in D; (nearest say in terms of distances)
2. Take class labels of the k nearest data points and apply majority vote; If majority of the nearest data points belong to positive class then the data point can be assigned with positive class label and vice versa;

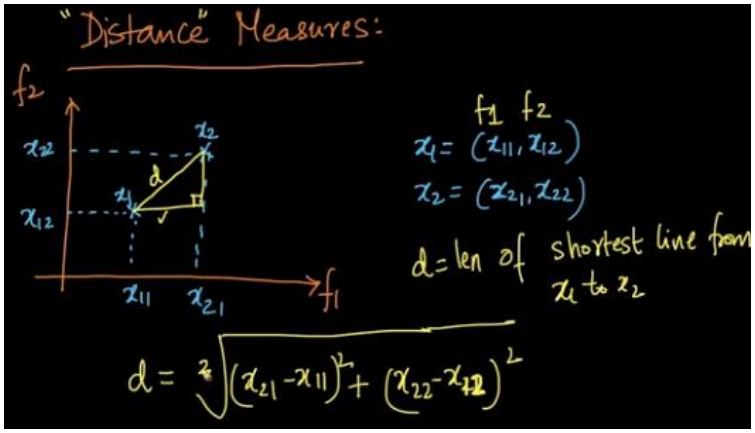
Get nearest neighbors and apply majority vote; If K is even then majority votes can result in ties, thus avoid even numbers for k

19.5 FAILURE CASES OF KNN

1. Outliers (nearest neighbors are far to the outlier query point, assigning a class based on kNN is not good)

2. If the dataset is excessively randomized or a completely mixed up data, then there is no useful information in the data; most ML models fail in this case;

19.6 DISTANCE MEASURES: EUCLIDEAN (L2) , MANHATTAN(L1), MINKOWSKI, HAMMING



This distance is known as a Euclidean distance (shortest distance between two points)

It is called as L2 norm when the input is a vector;

$x1 \varepsilon R^d$ and $x2 \varepsilon R^d$:

Euclidean distance: $||x1 - x2||_2 = (\text{summ}(x_{1i} - x_{2i})^2)^{1/2}$

$||x1 - x2||_2 \rightarrow$ L2 Norm of vector

$||x1||_2 = (\text{summ}(x_{1i})^2)^{1/2}$

Manhattan distance: $||x1 - x2||_1 = (\text{summ}(\text{abs}(x_{1i} - x_{2i})))$

$\rightarrow$ L1 norm of vector

$||x1||_1 = (\text{summ}(\text{abs}(x_{1i})))$

Minkowski distance: Lp norm: $||x1 - x2||_p = (\text{summ}(\text{abs}(x_{1i} - x_{2i}))^p)^{1/p}$

P>2

Distances are between two points and Norm is for vectors;

Hamming distance: Number of locations where there is a difference in two vectors

Text processing (binary BOW or Boolean vector)

$X_1 = [0, 1, 1, 0, 1, 0, 0]$

$X_2 = [1, 0, 1, 0, 1, 0, 1]$

H(X1, X2) = 1+1+0+0+0+0+1 = 3

X1 = 'abcadefghik'

X2 = 'acbadegfhik'

H(X1, X2) = 0+1+1+0+0+1+1+0+0+0 = 4

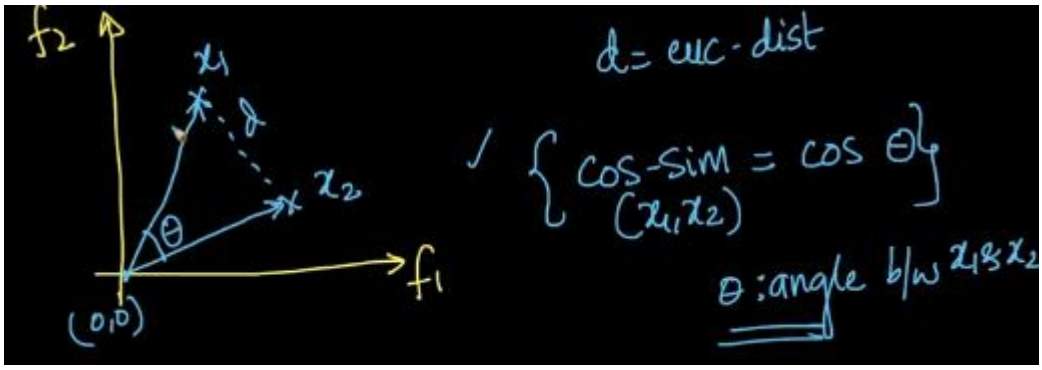
Hamming distance is applied to Gene encoding

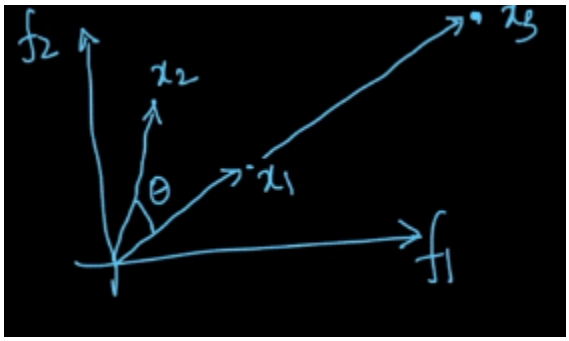
19.7 COSINE DISTANCE & COSINE SIMILARITY

Similarity vs Distance: As distance increases the two points are less similar;

Similarity and Distances have opposite behavior

cos_dist = 1 – cos_sim





cos_sim(x1, x3) = 1

cos_sim(x1, x2) = cos θ

cos_dist(x1, x3) = 1 – 1 = 0

cos_dist(x1, x2) = 1 – cos θ

Even though dist(x1, x3) > dist(x1, x2): cos_dist(x1, x3) < cos_dist(x1, x2);

Cos_sim(x1, x2) = x1.x2/$||x1||_2$ $||x2||_2$

Cosine similarity is related to Euclidean distance as follows. Denote Euclidean distance by the usual $\|A - B\|$, and observe that

$$\|A - B\|^2 = (A - B)^\top (A - B) = \|A\|^2 + \|B\|^2 - 2A^\top B$$

by expansion. When A and B are normalized to unit length, $\|A\|^2 = \|B\|^2 = 1$ so the previous is equal to

$$2(1 - \cos(A, B))$$

If x1 and x2 are unit vectors, then square of Euclidean distance between x1 and x2 = 2 times cosine distance between x1 and x2

19.8 HOW TO MEASURE THE EFFECTIVENESS OF K-NN?

Given: a text reviews data with class labels;

Problem: What is polarity (+ve, -ve) of a new review?

kNN → nearest neighbors + majority vote

To measure the performance of the any ML model:

Break the dataset into two sets train and test and there is no intersection between train and test data points (each data point either goes to train set or to test set)

Splitting can be done randomly (one of the methods) into train set = 70% of the total dataset and test set = rest 30% of the total dataset;

For each point in test dataset; make data point as a query point, then use train set and kNN model to determine y_q; then if class label is correct increment a counter by 1; for metrics we can use this counter which is the number correct classifications in the test set; accuracy can be computed by (counter/test size);

This accuracy will allow us to understand the performance or effectiveness of kNN model;

19.9 TEST/ EVALUATION TIME AND SPACE COMPLEXITY

Given a query point:

Input: Dtrain, k, query

Output: y_q

Knnpts = []

For each xi in dtrain: ~O(nd)

Compute d(xi, xq) → di ~ O(d)

Keep smallest k distances and store in a list; ~ O(1)

Majority vote: ~O(1)

Time complexity = O(nd)

Space: At evaluation:

Dtrain ~O(nd)

After deployment, Dtrain is required to be present in RAM;

19.10 KNN LIMITATIONS

1. Due to large space complexity, we will require large RAM at deployment;

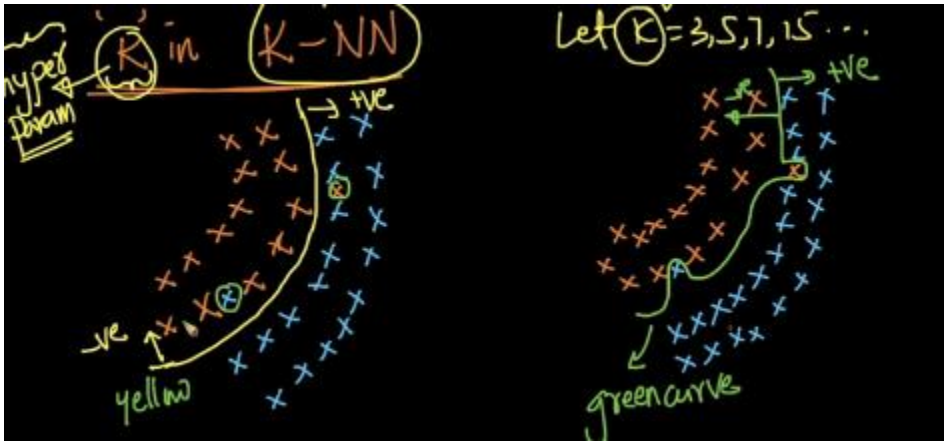
2. Low Latency is of high preference;

We can use kd tree and LSH for improving kNN performance;

19.11 DECISION SURFACE FOR K-NN AS K CHANGES

'k' in kNN is a hyper parameter;

k = 1;



Curves that separate +ve points from –ve points are called decision surfaces;

For k = 1:

Green curve is not smooth; and happens when k = 1;

For k = 5:

Yellow curve may be generated when k = 5; this is a smoother curve

As k increases smoothness of the decision surfaces increases;

For k = n:

All query points are classified as the majority class;

19.12 OVERFITTING AND UNDERFITTING

Overfitting: Generating extremely non smooth surfaces on training data to get extremely high accuracy on training set;

Underfit: Not generating any decision surface and classifying all query points with majority class;

Neither overfit nor underfit: Generating smooth surfaces that are less prone to noise;

19.13 NEED FOR CROSS VALIDATION

Determining k hyper parameter for kNN model:

We split the data set into train and test set;

We train the model using training set and compute accuracy using test set;

For every k we determine test accuracy and select k that gives best accuracy on test set;

Using test dataset to determine best k or best hyper parameters is not right;

Thus we split the data set into train, cross validate and test datasets; we want the model to have well generalization ability on future unseen data;

We train the model parameters compute nearest neighbors on training set, we determine best hyper parameters on cross validation dataset and we evaluate best ML models on test set; this will help ML models have good generalization ability;

19.14 K-FOLD CROSS VALIDATION

Data is split into train, cross validation and test set;

While test set should not be touched, cross validation data becomes untouched while training which will lead to loss of information;

We can use k-fold cross validation to incorporate cross validation data during training;

We are trying to get the information from cross validation set for ML model training;

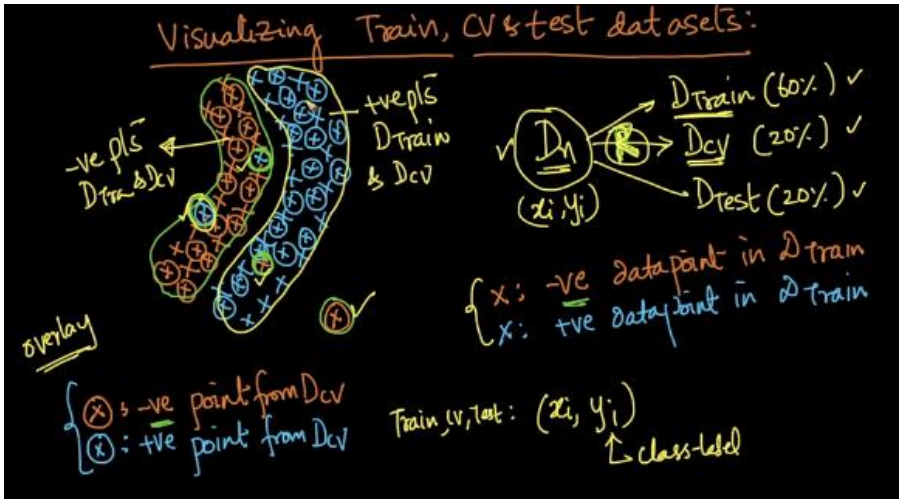
1. Dataset is split randomly into train and test set;
2. Train set is further divided randomly into k' equal sized parts;
3. For each k hyper parameter in kNN, we will use k'-1 parts of train set for training and the remaining 1 part of train set as cross validation data set, we then compute accuracy or model performance on cross validation data set; we will roll around the parts of train data set to get k' accuracies for k = 1 (kNN hyperparameter), we will then average this accuracies for k = 1; as a result we will have average of k' accuracies of the train set; this is called as k'-fold cross validation, we will repeat the k' fold cross validation for all hyper parameter k value choices;
4. We pick best k from best average k' cross validation accuracies;
5. And apply the best hyper parameter for measuring performance on test set;

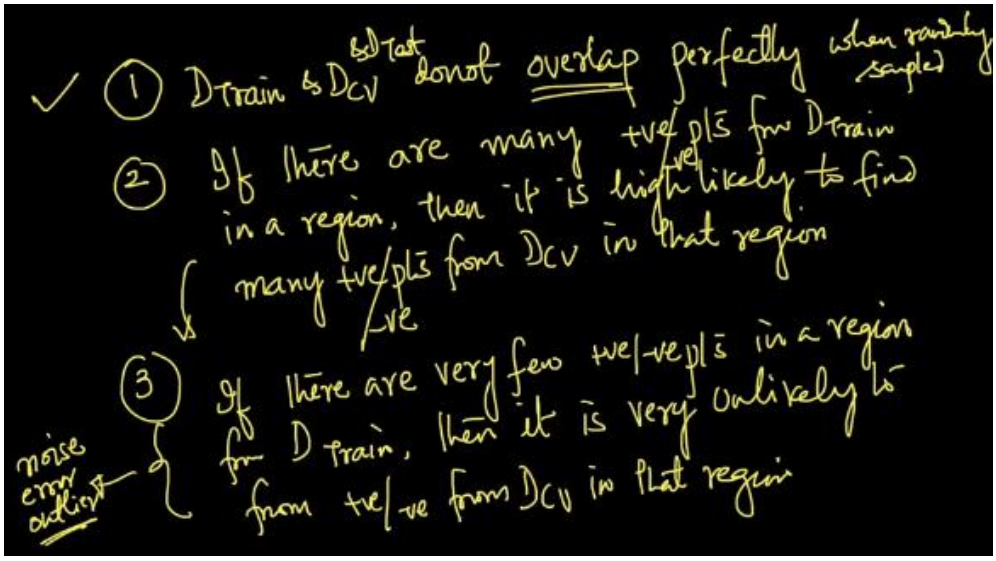
Generally 10 fold cross validation is applied;

With k' fold cross validation time taken for finding optimal k is multiplied by k' times;

But finding optimal k is a onetime effort;

19.15 VISUALIZING TRAIN, VALIDATION AND TEST DATASETS





19.16 HOW TO DETERMINE OVERFITTING AND UNDERFITTING?

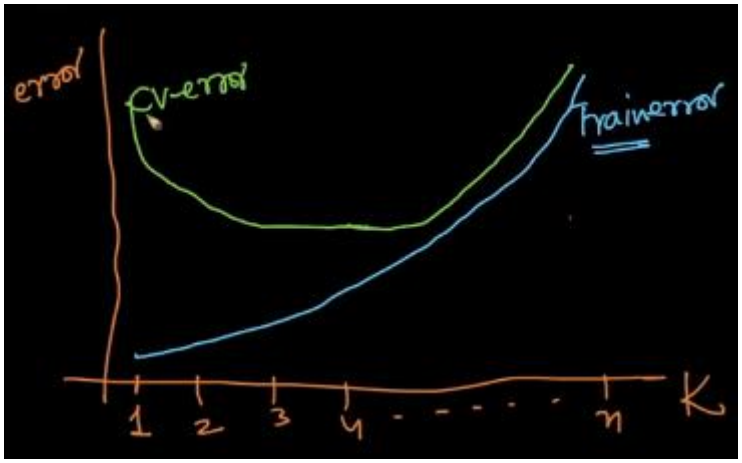
We are finding best k that is neither underfitting nor overfitting;

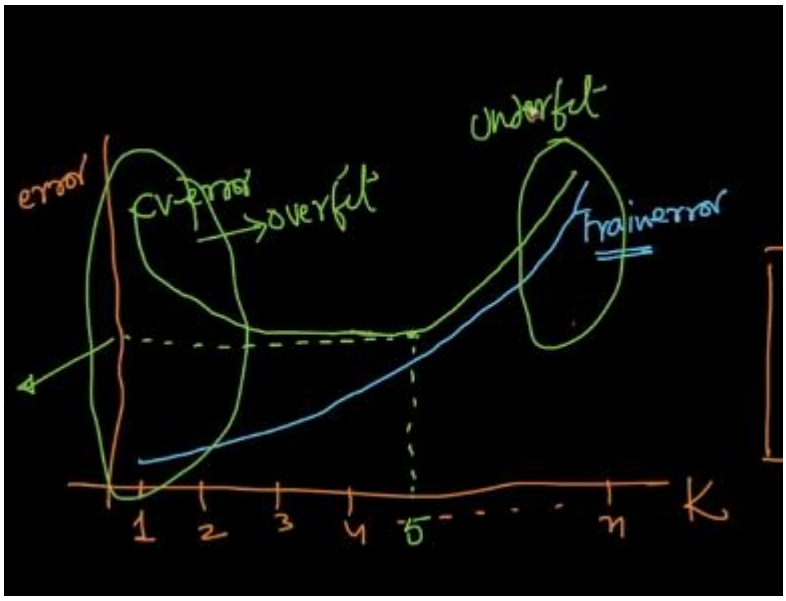
Accuracy: # correctly classified points/ total # of points

Error: 1 – accuracy

We can compute train error and cross validation error;

Plot a curve on error vs k hyperparameter;





If train error is high and validation error is high then we are underfitting;

If train error is low and validation error is high then we are overfitting;

At optimal k we will have closer train error and validation error;

19.17 TIME BASED SPLITTING

Data was split randomly so far;

For Amazon fine food reviews time based splitting is better as we also have timestamp feature in the dataset;

We will sort the data in the ascending order of timestamp;

If time stamps are available we need to check which of random split or time based split is better;

In random splitting: we will get reviews that were written before will occur in test set and the reviews that were written after will occur in train set;

In time based splitting, we will get reviews in train, cross validation and test set based on time stamps; the reviews that were written first occur in train set, then in cross validation and then in test set; the argument here is we should avoid predicting past data based on models trained on future datasets;

Reviews or products change over time; new products get added or some products get be terminated;

As time passes we require retraining the model with new data;

19.18 K-NN FOR REGRESSION

Classification: $D_n = \{(xi, yi) \mid xi \varepsilon R^d, yi \varepsilon \{0, 1\}\}$; we do majority vote

Regression: $D_n = \{(xi, yi) \mid xi \varepsilon R^d, yi \varepsilon R\}$; we will do mean or median of all k neighbors

19.19 WEIGHTED K-NN

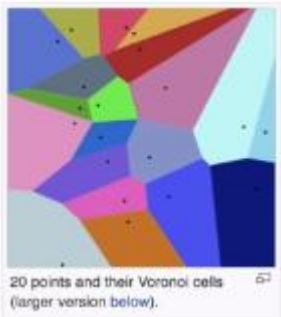
Weights depending on the distance of nearest neighbors;

Closer points will have high weights;

We can multiply the labels of nearest neighbors with reciprocal of the distance rather using simple majority vote;

19.20 VORONOI DIAGRAM

This divides the whole data space into regions based on nearest neighbor;



20 points and their Voronoi cells (larger version below).

19.21 BINARY SEARCH TREE

kNN: Time complexity: O(n) if d is small and k is small and Space complexity: O(n)

We can reduce time complexity to O(log n) using kd tree

Binary search tree: Given a sorted array, we can find presence of a number in the array in O(log n) time

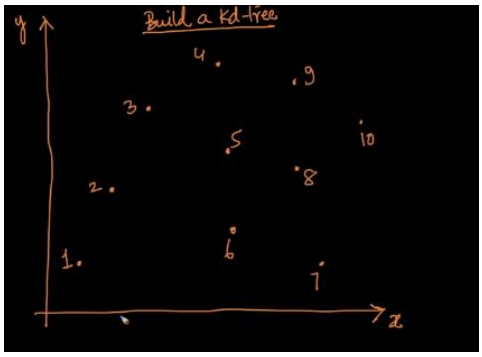
Construct a binary search tree; find element in array through comparisons;

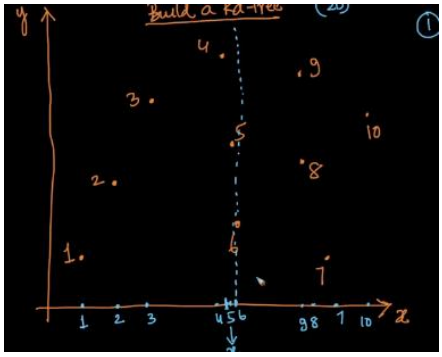
Depth of BST is O(log n)

19.22 HOW TO BUILD A KD-TREE

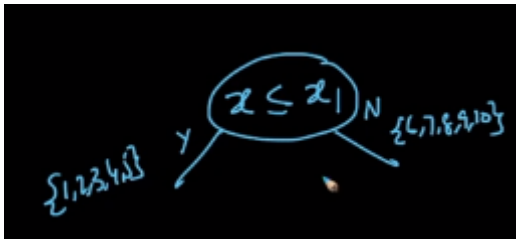
BST: Given a list of sorted numbers we can build a tree that divides the list in two at every stage

Kd – Tree:

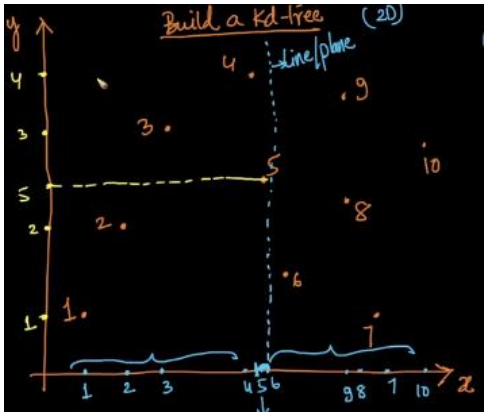


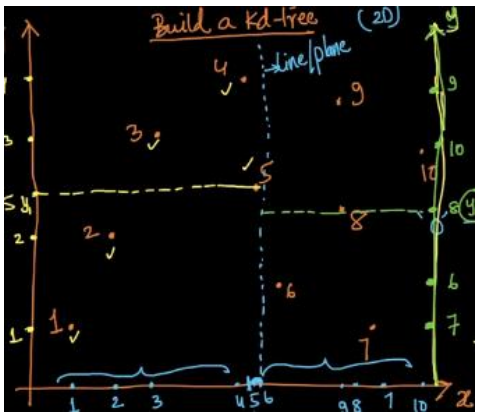


1. Pick first axis and project all data points on to the axis; pick the middle point; draw a plane through the middle point this divides the space into two equal halves;



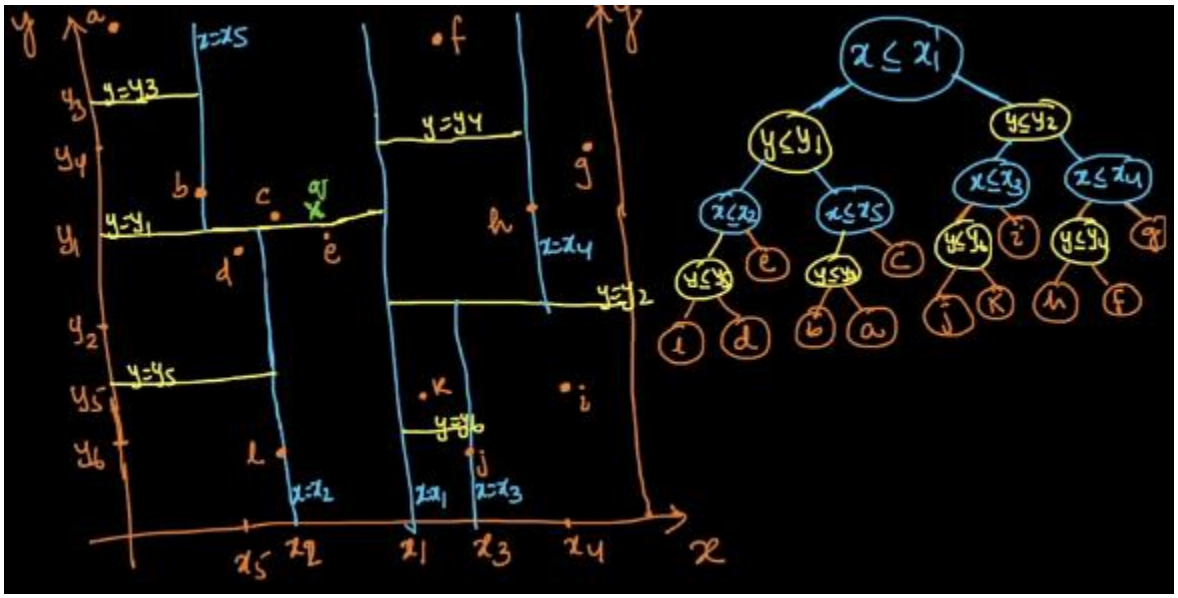
2. Change axis to y;





3. Change to x axis again and repeat steps; the data space is broken using axis parallel lines or hyper planes into hyper cuboids

19.23 FIND NEAREST NEIGHBOURS USING KD-TREE



xq <= x1, yq >= y1, xq >= x5 → c is one of the nearest points;

Draw a hyper sphere with centre xq and radius Dist(c,q); If there is another point is inside the hypersphere; this can also be checked as the hyper sphere intersects with y = y1 line;

We track back to y <= y1 condition in the tree and search for another nearest data point; now point e can also be a nearest neighbor;

Dist(q, e) < Dist(c,e): thus c is discarded as 1 nearest neighbor and take e as nearest neighbor; we repeat the steps done with c now with e; we can have a result that point e is the nearest neighbor to query point q;

Best case Time complexity is O(log n) and worst case t.c. is O(n);

For k Nearest Neighbors Time complexities: Best case O(k log n) and Worst case: O(kn)

19.24 LIMITATIONS OF KD TREE

As dimensionality increases the number of back tracks in kd tree worsens time complexity; kd tree nearest neighbor updates is exponentially proportional to dimensionality (2^d)

O(log n) is valid for uniform distribution and as distribution moves towards real world clusters, time complexity moves towards O(n);

19.25 EXTENSIONS

Go through Wikipedia article

19.26 HASHING VS LSH

Locality sensitive hashing to find nearest neighbors when d is large;

Given an unordered array apply a hash function;

> When we need to find an element in the array we can use the hash table to look up for key value that is equal to query element and the value in the hash table will give us indices;

Locality sensitive hashing: it computes a hash function such that nearest data points pairs are stored in the same bucket; for a new query point relevant bucket is searched for and k nearest neighbors are searched through data points in this bucket; this will reduce the need for searching throughout the data space;

19.27 LSH FOR COSINE SIMILARITY

cos_sim(x1, x2) = cos θ

As θ increases the cosine decreases; similarity decreases and distance increases;

LSH is a randomized algorithm; it will not give same answer every time; gives answers based on probability;

If two points are close in terms of angular distance, then the points are similar; these points should go to same bucket in hash table;

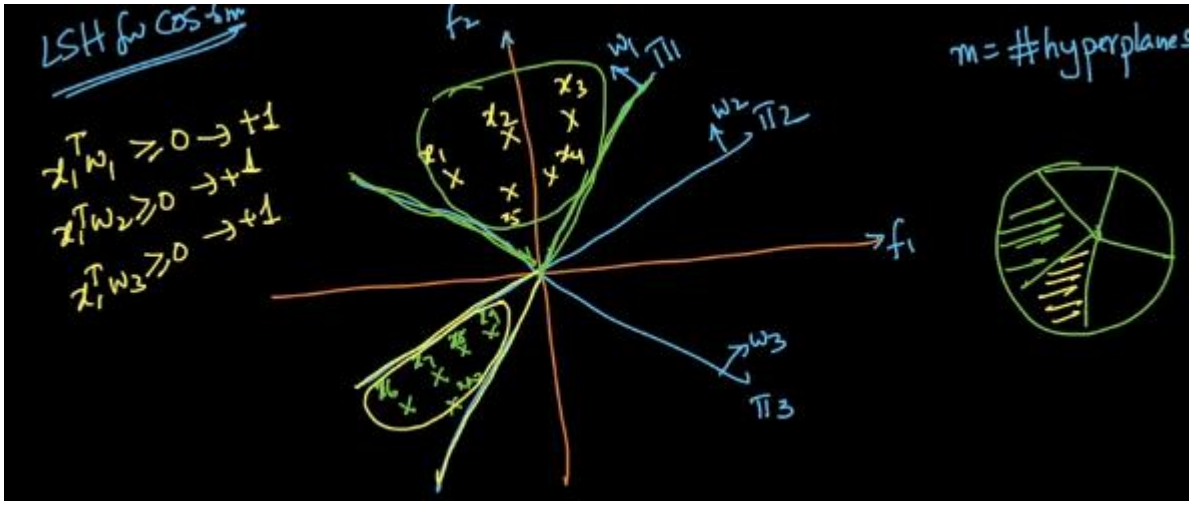
We break feature space using random hyper planes; for every plane we will have a unit normal vector 'w_i'

For plane 1: Say we have x1 above and x3 below the hyper planes; we will have:

$W_1^T . x1 >= 0$

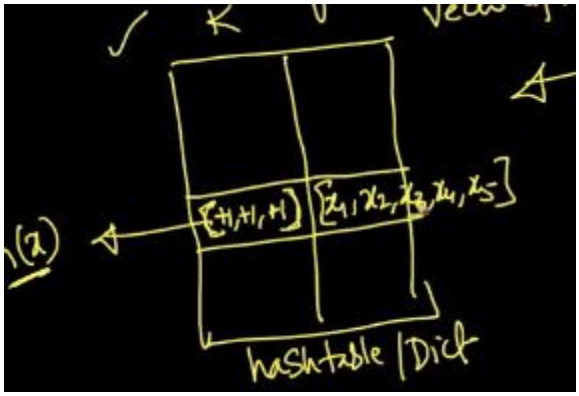
$W_1^T . x2 <= 0$

Random hyper planes are generated by generating a random W vector; each value in W vector is randomly generated from Normal distribution N(0,1) (0 mean and 1 variance);



H(x7) = [+1, -1, -1] (3 random hyper planes are generated)

The key will hash function value and all the data points will be stored in value bucket;



Time to construct the hash table: O(mdn) : m hyper planes; d dimensionality, n data points

Space O(nd)

Time at test time: O(md +n'd); n' is number of data points in hash table bucket

m is set roughly to log n;

LSH for cosine similarity can miss nearest neighbors when the nearest neighbor falls in opposite side to any hyper plane;

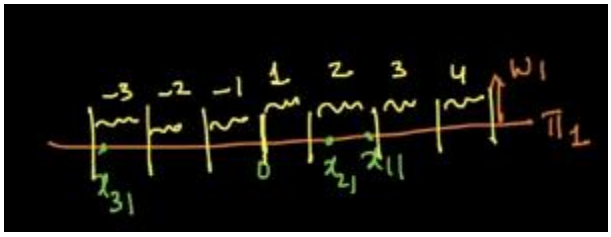
When iterating across LSH we can get nearest neighbors in a bucket in some iteration; over iterations at each bucket key of the hash table we can do union of all data points;

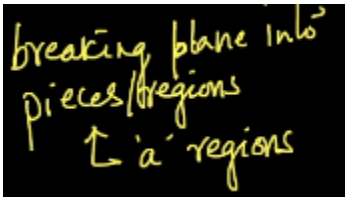
19.28 LSH FOR EUCLIDEAN DISTANCE

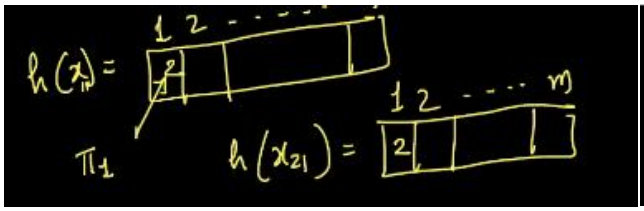
Simple extension:

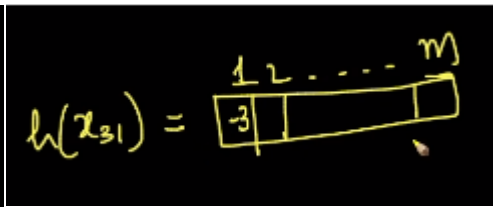
Each axis is divided into units:

Every point is projected on each axis of the data space; hash function will vectorize the data point into same dimensionality vector where each cell represents the part of the axis the data point projection lies on that axis;

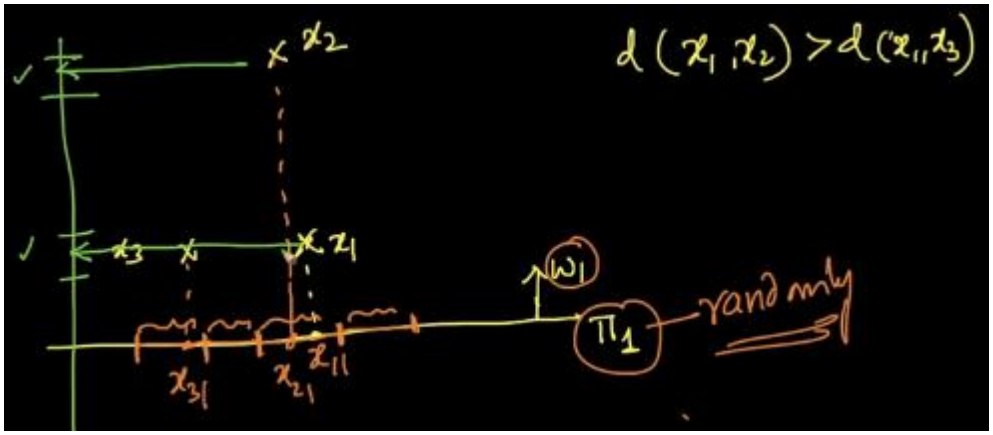








LSH for cosine similarity encodes each feature to +1 or -1;



On x axis x1 and x2 fall in same bucket, on y axis these are very far;

19.29 PROBABILISTIC CLASS LABEL

Say, with 3 NN we get a query data point prediction as positive and with 5 data points we can get the prediction as negative;

Also, say we go for 7NN; with a query data point it might happen that it nearest neighbors are 4 +ve and 3 –ve; say another query data point has 7 +ve and 0 –ve neighbors; then both the cases give prediction as +ve based on majority vote; but both data points do not have same number of +ve # of data points, query data point 1 is less +ve than query data point 2; thus giving a probabilistic quantification of the prediction will give us more confidence;

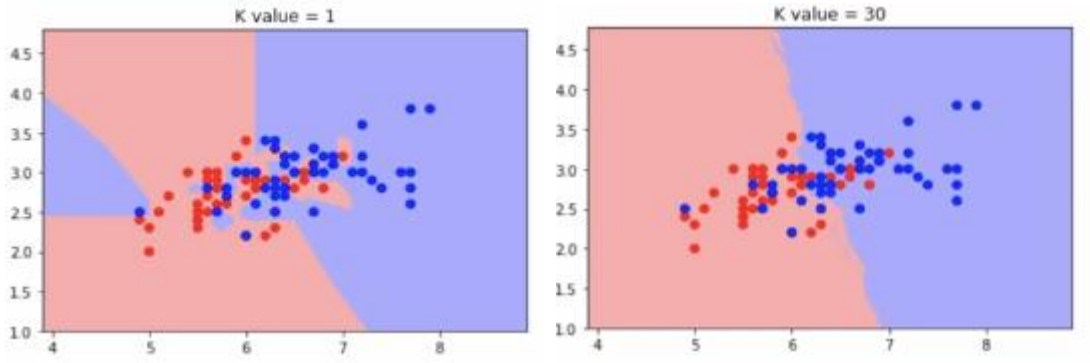
Query 1 has 4/7 +ve and 3/7 –ve;

Query 2 has 7/7 +ve and 0/7 –ve;

This gives certainty statistic of predictions;

19.30 CODE SAMPLE: DECISION BOUNDARY

sklearn.neighbors.KNeighbirsClassifier()



As k increases the decision boundary smoothens;

19.31 CODE SAMPLE: CROSS VALIDATION

Train test split()

19.32 REVISION QUESTIONS

1. Explain about K-Nearest Neighbors?
2. Failure cases of KNN?
3. Define Distance measures: Euclidean(L2) , Manhattan(L1), Minkowski, Hamming
4. What is Cosine Distance & Cosine Similarity?

5. How to measure the effectiveness of k-NN?
6. Limitations of KNN?
7. How to handle Overfitting and Underfitting in KNN?
8. Need for Cross validation?
9. What is K-fold cross validation?
10. What is Time based splitting?
11. Explain k-NN for regression?
12. Weighted k-NN ?
13. How to build a kd-tree.?
14. Find nearest neighbors using kd-tree
15. What is Locality sensitive Hashing (LSH)?(
16. Hashing vs LSH?
17. LSH for cosine similarity?
18. LSH for euclidean distance?

Chapter 20: INTERVIEW QUESTIONS ON K-NN (K NEAREST NEIGHBOUR)

20.1 QUESTIONS & ANSWERS

1. In k-means or kNN, we use euclidean distance to calculate the distance between nearest neighbours. Why not manhattan distance ?(https://www.analyticsvidhya.com/blog/2017/09/30-questions-test-k-nearest-neighbors-algorithm/)
2. How to test and know whether or not we have overfitting problem?
3. How is kNN different from k-means clustering?(https://stats.stackexchange.com/questions/56500/what-are-the-main-differences-between-k-means-and-k-nearest-neighbours)
4. Can you explain the difference between a Test Set and a Validation Set?(https://stackoverflow.com/questions/2976452/whats-is-the-difference-between-train-validation-and-test-set-in-neural-netwo)
5. How can you avoid overfitting in KNN?

External Resources: 1.https://www.analyticsvidhya.com/blog/2017/09/30-questions-test-k-nearest-neighbors-algorithm/

Chapter 21: CLASSIFICATION ALGORITHMS IN VARIOUS SITUATIONS

21.1 INTRODUCTION

Applying any classification algorithms;

Rather than knowing multiple algorithms; it is important to know type of problems that arise in real world;

21.2 IMBALANCED VS BALANCED DATASET

2 classes: Dn: n data points: n1 +ve, n2 –ve pts; n1 + n2 = n

If n1 ~ n2 → balanced dataset (n1:n2 = 58:42)

If n1<<n2 → imbalanced dataset (n1:n2 = 5:95); the results will be biased towards majority class;

Given imbalanced dataset:

1. Undersampling: n1 = 100 and n2 = 900

   Create new dataset with 100 n1 points and 100 n2 points randomly selected; result 100 n1 and 100 n2 points;

   We are discarding valuable information; 80% of the dataset is discarded

2. Oversampling: n1 = 100 and n2 = 900

   Create new dataset with 900 n1 points by repeating each point 9 times; and 900 n2 points; repeating more points from minority class to make the dataset a balanced dataset;

   We can create artificial or synthetic new points through extrapolation to increase n1 from 100 to 900;

   We are not losing any data; we can also give weights to classes; more weight to minority class;

   The nearest data point if belongs to minority class it is counted as 9 points;

When directly using the original imbalanced dataset; we can get high accuracy with a dumb model that predicts every query point to belong to majority class;

21.3 MULTI-CLASS CLASSIFICATION

Binary classifier: $y \in \{0, 1\}$

Multi class classifier:

MNIST: $y \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

kNN uses majority vote, it can easily be extended to multi class classification very easily;

But can we convert a multi class classification into a binary classifier;

The dataset is broken into parts such that for the first classifier we will have dataset with class labels class1 present and class1 absent;

We will build c binary classifiers for multi class classification problem with c labels;

This is called as One versus rest technique to do multi class classification;

21.4 K-NN, GIVEN A DISTANCE OR SIMILARITY MATRIX

If data points cannot be vectorized;

But say we can get similarity matrix between data points;

Example: similarity between medicines;

We will have rows and column with indices as data points and cell values will be similarity between row data point and column data point; we can convert similarity matrix into distance matrix using reciprocal of each similarity values;

We can use kNN for this distance or similarity matrices as kNN works on distances;

Techniques like Logistic regression on other side require d dimensional vector; thus we can think of limitations of various algorithms;

21.5 TRAIN AND TEST SET DIFFERENCES

Given a dataset we had random splitting or time based splitting;

We can have new products getting added after some time or old products removed;

The distribution of data points will change a lot;

Train and CV error can be low; but we can have test error high;

We need to check the distribution of train and test data sets for any change over time;

D_n – D_{train} (xi, yi) – (xi', yi')

D_{test} (xi, yi) – (xi', yi')

D_n' -- D_{train} ((xi, yi), 1)

D_{test} ((xi, yi), 0)

Train a binary classifier on Dn'; if the accuracy is high then Dtrain and Dtest are dis-similar; we will have small accuracy if Dtrain and Dtest are similar; to get Dtrain and Dtest follow same distribution (stable data) we will need to get small accuracy;

21.6 IMPACT OF OUTLIERS

kNN is highly impacted by outliers when k is small; decision surfaces change due to outliers;

Larger k is preferred when their performances are similar;

Outliers can be detected and removed;

21.7 LOCAL OUTLIER FACTOR (SIMPLE SOLUTION: MEAN DISTANCE TO KNN)

To detect outliers in data inspired by ideas in kNN;

We can have different dense clusters in the data space;

For every data point compute get k Nearest Neighbors, compute average of k distances; sort data points with by average distances; this will remove global outliers;

Compute locate density; to remove local outliers;

21.8 K DISTANCE

K_distance (xi) = distance of the kth nearest neighbor of xi from xi;

N(xi) = Neighborhood of xi; set of nearest neighbors

21.9 REACHABILITY-DISTANCE (A,B)

Reachability_distance(xi, xj) = max(k_distance(xj), dist(xi, xj))

If xi is in N(xj): then reachability_distance(xi, xj) = k_distance(xj)

Else: reachability_distance(xi, xj) = dist(xi, xj)

21.10 LOCAL REACHABILITY-DENSITY (A)

LRD(xi) = 1/(summ (reach_dist(xi, xj)/|N(xi)|)

Summation over all the points in the neighborhood of reachability distance by number of elements in the neighborhood;

Inverse of average reachability distance of xi from its neighbors;

21.11 LOCAL OUTLIER FACTOR (A)0

LOF(xi) = (summ(lrd(xj)/|N(xi)| * 1/ lrd(xi)

= average lrd of pts in the neighborhood of xi / lrd of xi

If LOF(xi) is large, then xi is an outlier; it is large when lrd(xi) is small compared to its neighbors, that is the density near the point is small compared to its nearest neighbors;

If the density is small then the point is outlier;

Sklearn.neighbors.LocalOutlierFactor()

21.12 IMPACT OF SCALE & COLUMN STANDARDIZATION

Features in a dataset can have different scales; the distance measures will not be proper; the features that have large scales dominate the distance measurements;

Example:

X1 = [23, 0.2]

X2 = [28, 0.2]

X3 = [23, 1.0]

Dist(X1, X2) = 5;

Dist(X1, X3) = 0.8;

% difference in X1 and X2 = 0.05 and X1 and X3 = 0.8, though X1 and X3 are far compared to X1 and X2, Euclidean distance(X1, X2) < E. d. (X1, X3)

Since Euclidean distance can be impacted by scale; column standardization should be applied before computing distances;

21.13 INTERPRETABILITY

A machine learning model should be interpretable; in addition to giving predictions it should explain why a prediction is made; this will make decisions reliable;

Example: Cancer prediction:

> An ML model trained on cancer data can provide predictions whether a new patient has cancer or not; but we completely do not rely on machines as this will cause chaos in real life; we will have a doctor who reads the predictions from the ML model and then communicates the findings with the patient; in addition to predictions the doctor needs evidence in terms of test results and other requirements to be provided by the ML model; based on the test results the doctor can explain the patient the presence or absence of cancer;
>
> Such a model is called an interpretable model;
>
> A model that does not give reasoning or does not provide easy access to its decisions or predictions computation is called a black box model;
>
> kNN is an interpretable model; as it can show the nearest data points based on which it has made prediction; the doctor can read similar patient records and come to a conclusion whether a patient has cancer or not;
>
> The data point vector can be comprised of results of medical tests; such as weight, blood group, etc.

21.14 FEATURE IMPORTANCE AND FORWARD FEATURE SELECTION

Important features: Features those are useful for the machine learning model in making predictions; this improves model interpretability; feature importance allows us understand a model better;

kNN does not give feature importance by default;

Forward feature selection: Given a model f through forward feature selection we can use the model itself to get feature importance; Given a high dimensionality dataset we want to reduce dimensionality to make things easier for computations (curse of dimensionality); One way is to use PCA/ tSNE; but PCA and tSNE care about distances and do not care about classification task; but for classification task using forward feature selection we can discard less important features;

1. Given a dataset of d features; use each feature at a time to train an ML model, the performance of the model is noted with respect to each feature; the feature that gave highest accuracy is selected say this is fs1 (feature selected at stage1)
2. Retrain the model with remaining features in concatenation with fs1 one at a time; we will get fs2; here fs2 + fs1 will give highest accuracy;
3. Repeat these steps up to fsd; these stage wise concatenation of features to gain high performance from the model is called forward feature selection;

Note: At first stage we have the second best feature; it may happen that this feature in combination with the first best feature may not provide good performance; so at each stage we check that given that a feature or set of features are selected previously as best features we now explore for features that add most value to model performance;

We can have a backward selection; where we try to remove the feature that results in lowest drop in performance of the mode;

At any iteration we are training the ML model; time complexity is very high;

21.15 HANDLING CATEGORICAL AND NUMERICAL FEATURES

Given a dataset we can have multiple feature data types;

For a height prediction regression problem, we have weight, country, hair color, etc as features; Weight is a numerical feature which can be given an input to the model; Hair color and country are categorical text input; ML models take numerical as input; we require to convert text and categorical features into equivalent numerical feature;

1. Say for hair color we have black, red, brown, Gray as values; we can assign numbers to each color as black = 1; red = 2, etc; but numbers are said to be ordinal; 2 is greater than 1 but red greater than black is absurd; with this numerical conversion we are inducing an artificial order in the categories;
2. Thus one hot encoding is a better option; each category in the color feature is made as a new binary feature; disadvantage: dimensionality of the dataset increases and the dataset will be a sparse matrix as in each row there will only one cell which is non zero;
3. Mean replacement category wise: Replace country column by the average height of the people from that country;
4. Using domain knowledge: example for country we can replace the value with distance from some reference country; or coordinate location on map; say we have some fact that person near equator are tall and person away from equator is short which is stated by a domain expert;

Ordinal features such as ratings: we can convert that into numbers which can be compared (V. good, good, avg, bad, v. bad); = (5, 4, 3, 2, 1) or (10, 6, 4, 2, 1) decisions on this are random;

21.16 HANDLING MISSING VALUES BY IMPUTATION

Given a dataset Dn, we could have missing values due to many reasonable reasons;

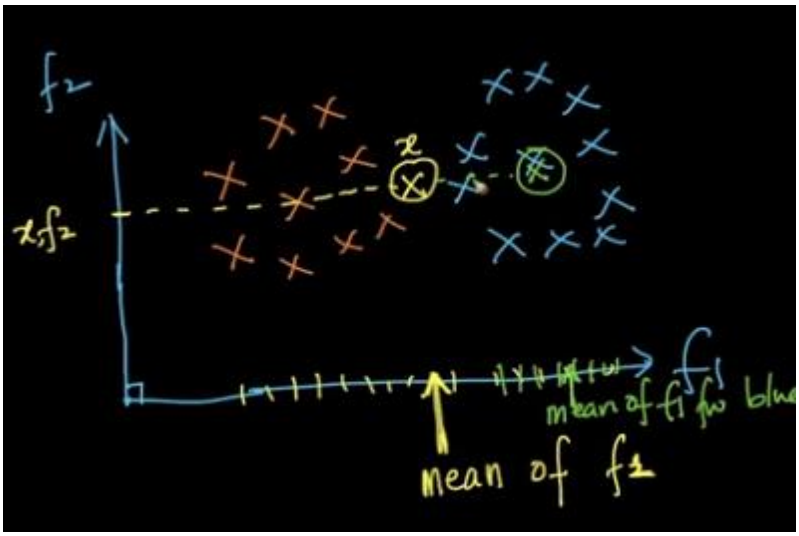
Task: Filling missing values:

1. Mean imputation, median imputation and mode imputation;

   We can impute missing values with mean, median or mode of the feature;

2. Classification task: we can also impute based on class labels;

   We can have Mean of all points or mean of all positive points or negative points;



   Yellow mean of whole dataset; Green: mean of blue points only;

3. In addition to imputation we can add a new missing feature: we can create a new feature which also tells whether a missing value appeared in the original feature; we are adding binary features which tell us that a value is missing
4. Model based imputation: We will consider feature that has missing values as target variable and make predictions based on non missing features; Missing values are kept into test set; kNN is used for model based imputation where neighborhood can be used to fill or impute missing values;

21.17 CURSE OF DIMENSIONALITY

When d is high:

- In machine learning; let all features be binary where 2 is minimum possible number of values for the feature0 and 1; for d number of features we will have 2^d possible values for the dataset; so as dimensionality increases the number of data points required for the model to perform well increases exponentially;

  Hughes phenomenon: for fixed number of training samples the predictive power reduces as the dimensionality increases;

  As dimensionality increases the number of data points required for the model to perform well increases incrementally;

- Distance functions: intuition of distances becomes invalid in high dimensionality;

  If a large number of random data points are selected in a high dimensionality space, the minimum of distance between every pair of points is roughly similar to the maximum of distance between every pair of points; this makes comparison of Euclidean distances for data points similarity impossible as dimensionality increases;

  Cosine similarity can be considered instead of Euclidean distance though cosine similarity is impacted by curse of dimensionality the impact is less when compared with the impact on distances;

  For high dimensional data, the impact of curse of dimensionality is high for dense data compared to the impact on sparse data;

- Overfitting and Underfitting: As dimensionality increases chances of overfitting increases; we can avoid this with forward feature selection; we can also use PCA or tSNE which are not classification oriented, these preserve proximity and variance;

kNN on text data: cosine similarity and sparse representations as these are less impacted;

Additionally:

You can also consider the length of diagonal for a unit hyper cube:

For 2D L(dia) = sqrt(2)

For 3D L(dia) = sqrt(3)

For 10 000 D L(dia) = sqrt(10000) = 100

Thus even though the data points lie in the same unit hyper cube they are very far;

Similarly the average distance of random data points in a nD hyper cube is given as sqrt(10^n/n);

For 2D: avg dist = sqrt(100/2) = 7.07

For 3D: 18.25

For 6D: 408.24

Thus the average distance you can get from a 6D dataset is very large; and real world problems come with dimensionality in range of 100s and 1000s, which worsens the distance measurements and the intuition of similarity among data points;

21.18 BIAS-VARIANCE TRADEOFF

In kNN we saw for k = n we have underfitting and for k = 1 we have overfitting;

Generalization error = error ML model makes on future unseen data

Generalization Error = Bias^2 + Variance + irreducible error

Bias: due to simplifying assumptions; example for a quadratic function if we assume that the data points follow a linear function we will have bias;

Example: if we have an imbalanced dataset and if we assume that the whole dataset belongs to dominant class we will have bias;

High bias = underfitting due to simplifying assumptions;

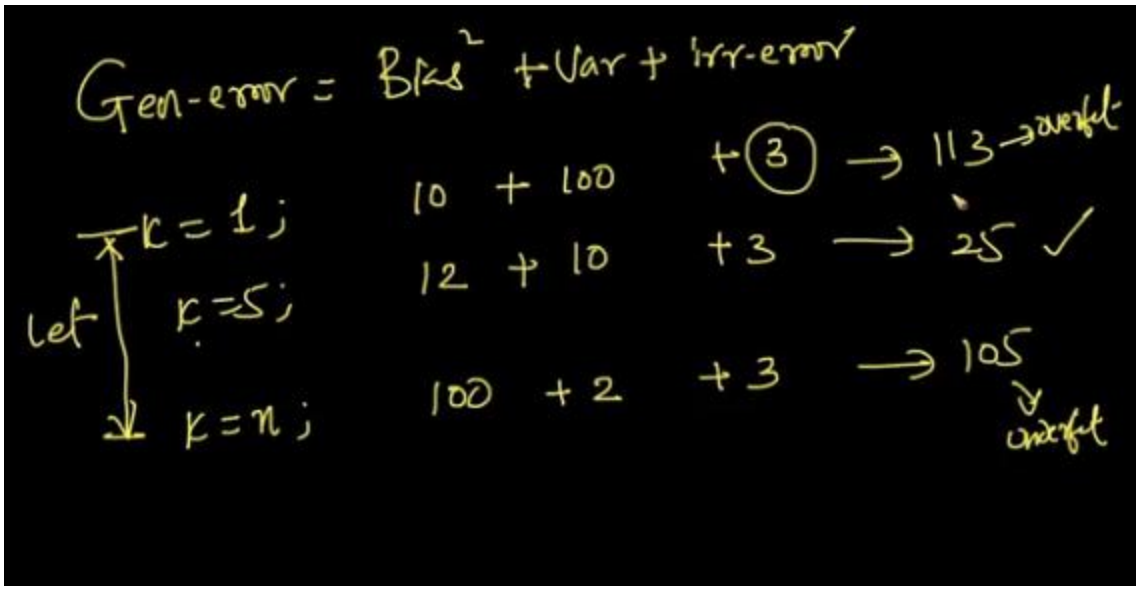
Variance: how much a model changes as training data changes; if the model does not change much with changes in training data we will have a low variance model ;

High bias = overfitting due to restrictions;

For k = 1 in kNN small changes in dataset result in large changes in decision surfaces; this model will have high variance; As k increases variance reduces;

At k = 1 we will have a low bias and high variance model; as k increases bias increases slightly while variance increases drastically;

High bias leads to underfitting and high variance leads to overfitting thus we need a balance between bias and variance; as variance increases bias decreases;



21.19 INTUITIVE UNDERSTANDING OF BIAS-VARIANCE

Given dataset D split into Dtrain and Dtest; a model is built on train data; We will have train error and test error;

If train error is high we will have high bias which means the model is underfitting; if train error is low then bias will be low

For bias: we should have low train error;

For variance: we should have similar train and test error;

If train error is low and test error is high we will have model overfitting on train data; this model will have high variance; we can also observe for changes in the model predictions due to changes in training data;

21.20 REVISION QUESTIONS

1. What is Imbalanced and balanced dataset.
2. Define Multi-class classification?
3. Explain Impact of Outliers?
4. What is Local Outlier Factor?
5. What is k-distance (A), N(A)

6. Define reachability-distance(A, B)?
7. What is Local-reachability-density(A)?
8. Define LOF(A)?
9. Impact of Scale & Column standardization?
10. What is Interpretability?
11. Handling categorical and numerical features?
12. Handling missing values by imputation?
13. Bias-Variance tradeoff?

21.21 BEST AND WORST CASE OF ALGORITHM

kNN:

1. When dimensionality is small: kNN is a good algorithm; for high dimensionality we will have curse of dimensionality and interpretability reduces; we will high run time complexity;
2. At run time we will have high time complexity; kNN cannot be used for low latency applications;
3. If we have the right distance measure, then kNN can be applied; if a right distance measure is not known kNN should be avoided; kNN is very good for similarity matrix or distance matrix;

Chapter 22: PERFORMANCE MEASUREMENT OF MODELS

22.1 ACCURACY

Task: Measure performance of ML models;

Accuracy: ratio of number of correctly classified points to total number of points;

Case 1: if we have an imbalanced dataset, with a dumb model we can get high accuracies; Accuracy cannot be used for imbalanced datasets;

Case 2: If we have models which return a probability score; an inferior model predicts probability values far from true labels (near 0.5 predictions) while a powerful model predicts probability values nearer to true labels (far from 0.5 predictions); accuracy can say that an inferior model is working similar to a powerful model; thus accuracy cannot give an idea whether a model is inferior or powerful;

22.2 CONFUSION MATRIX, TPR, FPR, FNR, TNR

In a binary classification problem we have two classes; a confusion matrix can be built with column vectors as predicted class labels and row vectors as actual class labels; Confusion matrix cannot process probability values;

Confusion Matrix	Actual 0	Actual 1
Predicted 0	a	b
Predicted 1	c	d

Confusion Matrix	Predicted 0	Predicted 1
Actual 0	a	c
Actual 1	b	d

To construct a confusion matrix we need to have true class labels and predicted class labels;

a = # of points that have actual and predicted labels as 0;

For Multi class classification: we will have a matrix of size cxc where c is the number of classes;

If the model is sensible, then the principal diagonal elements will be high and off diagonal elements will be low;

a = True Negatives, b = False Negatives, c = False Positives, d = True Positives

True Negatives: Predicted Negatives that are true or correct

False Negatives: Predicted Negatives that are false, these points are actually positives

True Positives: Predicted Positives correctly;

False Positives: Predicted Positives are wrong, these are actually negatives;

TPR (True Positive Rate) = TP / P = TP/ (TP+FN)

TNR = TN/N = TN/ (TN + FP)

FPR = FP / N

FNR = FN / P

Model is good if TPR, TNR are high and FPR, FNR are low;

A dumb model makes one of TPR or TNR = 0;

The condition for which of TPR, TNR, FPR and FNR should be considered depends on domain;

For medical purposes: we don't want to miss a patient who has a disease; rather a patient with no disease can further be sent to tests; but a patient cannot be left untreated due to False Negatives;

22.3 PRECISION AND RECALL, F1-SCORE

Precision = TP/ (TP + FP); of all the predicted positives how many are actually positive

Recall = TP / (TP + FN); of all actual positives how many are predicted positive

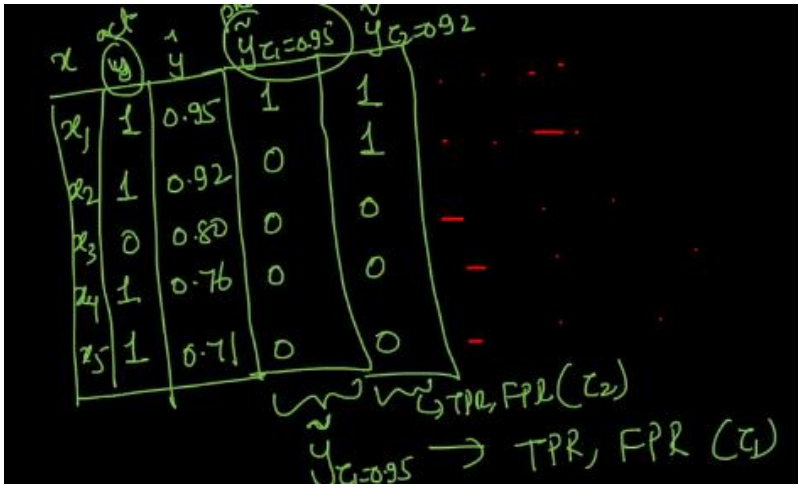
F1 score = 2 * Pr * Re / (Pr + Re); high precision will result in high precision and high recall; f1 score is harmonic mean of precision and recall;

F1 score = 2/(1/PR + 1/Re) = 2TP/(2TP + FP + FN)

22.4 RECEIVER OPERATING CHARACTERISTIC CURVE (ROC) CURVE AND AUC

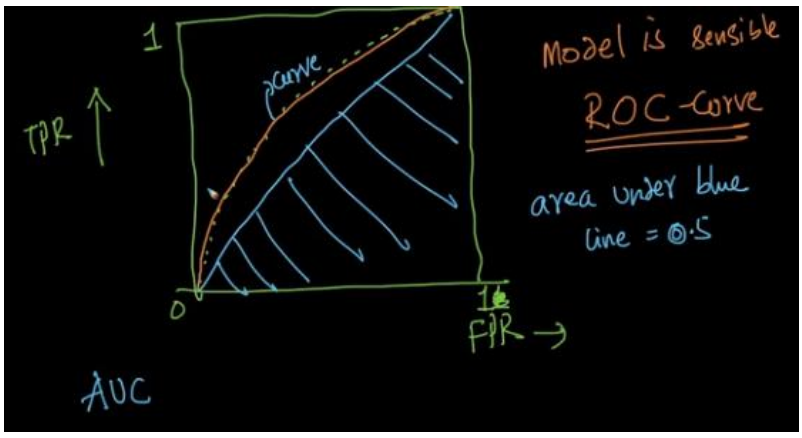
Sort the data frame by descending order of predicted class labels;

Thresholding:



We can have n thresholds with n TPR and FPR;

These TPR and FPR can be plotted which generates a curve called Receiver Operating Characteristic Curve;



AUC ranges from 0 to 1; 1 being ideal; AUC can be high even for a dumb model when the data set is imbalanced;

AUC is not dependent on the predicted values, rather it considers the ordering; if two models give same order of predicted values then AUC will be same for both the models;

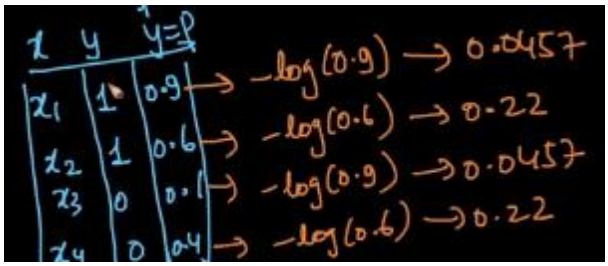
AUC for a random model is 0.5;

Preferred AUC is a value > 0.5; AUC = 0.5 for a random model and AUC between 0 and 0.5 imply that the predictions are reversed; if the predictions are again reversed then the new AUC value will be 1 – old AUC;

22.5 LOG-LOSS

Binary classification problem: Log Loss uses probability scores;

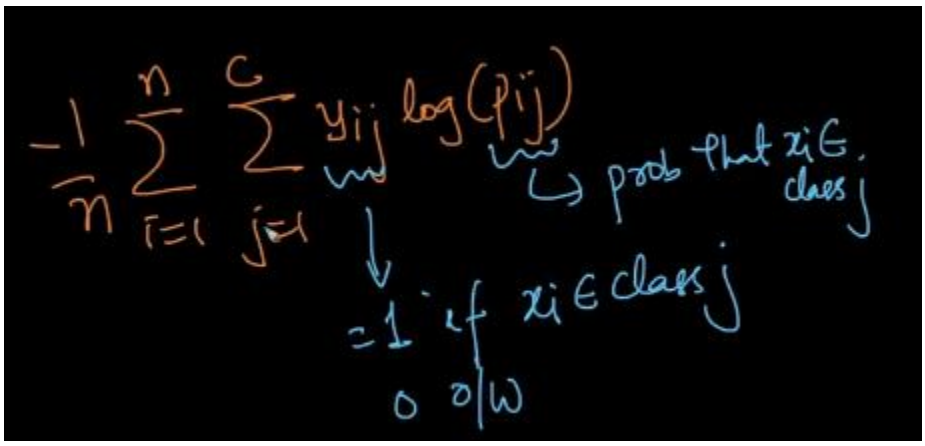
Log loss = $- (1/n) \text{ summ}(y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i)$



Log loss takes care of mis-classifications; this metric penalizes even for small deviations from actual class label

Log – loss: average negative logarithm of probability of correct class label;

For multi class classification:



22.6 R-SQUARED/COEFFICIENT OF DETERMINATION

Sum of squared errors: sum of $(y_i - y_{avg})^2$;

A simplest model can output for every query point a mean of the whole dataset;

Sum of squares of residues = sum $(y_i - y_{pred})^2$;

$R^2 = 1 - (SS_{res} / SS_{tot})$;

Case 1: $SS_{res} = 0$; $R^2 = 1$ – best value

Case 2: $SS_{res} < SS_{tot}$; $R^2 = 0$ to 1;

Case 3: $SS_{res} = SS_{tot}$; $R^2 = 0$: this model is same as simple mean model

Case 4: $SS_{res} > SS_{tot}$; $R^2 < 0$: this model is worse than a simple mean model

22.7 MEDIAN ABSOLUTE DEVIATION (MAD)

SS_{res} = summ (e_i^2); if one of the errors is very large, then R^2 is highly impacted, it is not robust to outliers;

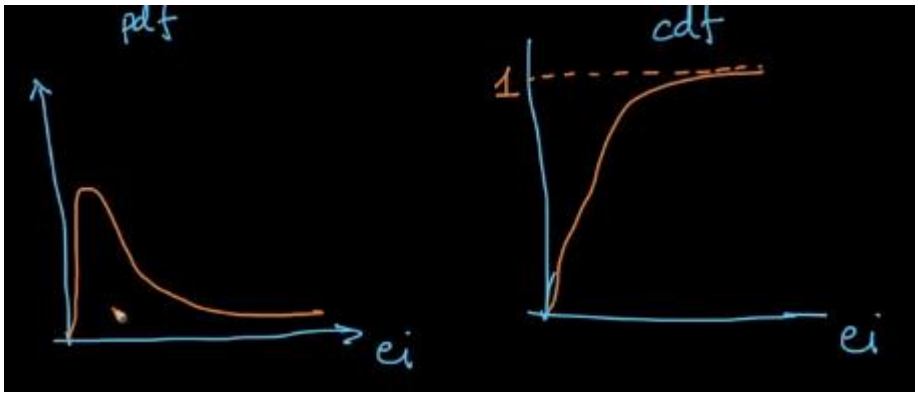
Median Absolute Deviation is robust metric: Median ($|e_i - \text{Median}(e_i)|$)

This is a robust measure of standard deviation;

Median(e_i) = central value of errors;

22.8 DISTRIBUTION OF ERRORS

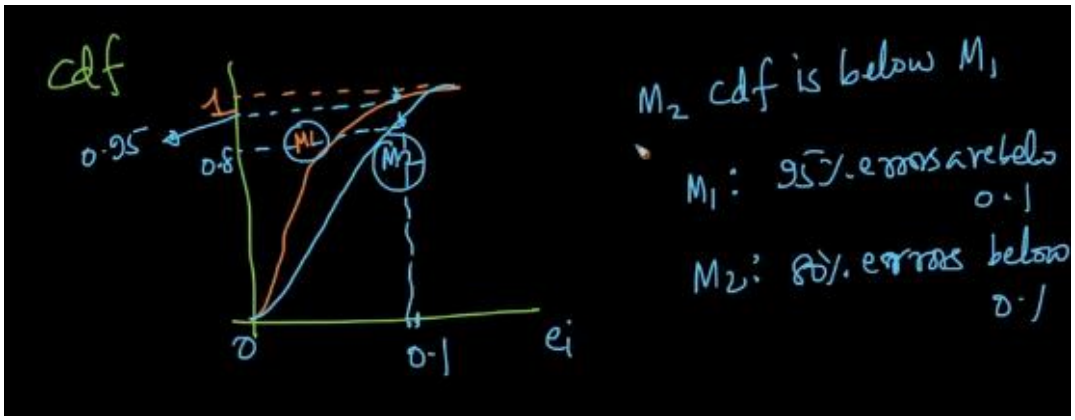
If we plot PDF and CDF for errors (e_i):



Very few errors are large; ideally we require 0 errors; from CDF we can get the percentage of data points that have errors;

This can be used to compare two models;

If model M1 CDF is above M2 CDF then model M1 is better than model M2



22.9 REVISION QUESTIONS

1. What is Accuracy?
2. Explain about Confusion matrix, TPR, FPR, FNR, and TNR?
3. What do you understand about Precision & recall, F1-score? How would you use it?
4. What is the ROC Curve and what is AUC (a.k.a. AUROC)?
5. What is Log-loss and how it helps to improve performance?
6. Explain about R-Squared/ Coefficient of determination
7. Explain about Median absolute deviation (MAD)? Importance of MAD?
8. Define Distribution of errors?

Chapter 23: INTERVIEW QUESTIONS ON PERFORMANCE MEASUREMENT MODELS

23.1 QUESTIONS & ANSWERS

1. Which is more important to you– model accuracy, or model performance?
2. Can you cite some examples where a false positive is important than a false negative?
3. Can you cite some examples where a false negative important than a false positive?
4. Can you cite some examples where both false positive and false negatives are equally important?
5. What is the most frequent metric to assess model accuracy for classification problems?
6. Why is Area Under ROC Curve (AUROC) better than raw accuracy as an out-of- sample evaluation metric?

Chapter 24: NAIVE BAYES

24.1 CONDITIONAL PROBABILITY

A classification algorithm based on probability;

kNN: Neighborhood based classification or regression model;

P(A|B) = P(A) * P(B|A) / P(B)

Example: 2 fair 6 sided dice:

D1: Rolling dice 1; D2: Rolling dice 2;

Sample space = 36 outcomes

P(D1 = 2) = 1/6

P(D2 = 2|D1 = 2) = 1/36;

P(D1+D2 <=5) = 10/36

+		D2					
		1	2	3	4	5	6
D1	1	2	3	4	5	6	7
	2	3	4	5	6	7	8
	3	4	5	6	7	8	9
	4	5	6	7	8	9	10
	5	6	7	8	9	10	11
	6	7	8	9	10	11	12

P(D1 = 2 | D1+D2 < = 5) = 3/10

Table 3

+		D2					
		1	2	3	4	5	6
D1	1	2	3	4	5	6	7
	2	3	4	5	6	7	8
	3	4	5	6	7	8	9
	4	5	6	7	8	9	10
	5	6	7	8	9	10	11
	6	7	8	9	10	11	12

There are 10 outcomes where D1 + D2 <= 5 and out of these 10 outcomes D1 = 2 for 3 outcomes

P(A|B) = P(A and B) / P(B);

P(A and B) = P(A intersection B) = P(A|B) * P(B) = P(B|A)*P(A)

24.2 INDEPENDENT VS MUTUALLY EXCLUSIVE EVENTS

P(A and B) = P(A intersection B)

Independent events:

P(A|B) = P(A) and P(B|A) = P(B);

Mutually exclusive events:

P(A|B) = P(B|A) = 0;

P(A intersection B) = P(B intersection A) = 0

24.3 BAYES THEOREM WITH EXAMPLES

P(A|B) = P(B|A) * P(A) / P(B) if P(B) !=0

Posterior = likelihood*prior/evidence

P(A|B) = P(A and B) / P(B) = P(A, B) / P(B)

P(A, B) = P(B, A)

P(B|A) = P(B, A)/P(A)

P(B,A) = P(B|A) * P(A) = P(A, B) = P(A|B) * P(B)

Example:

Output: M1: 0.2, M2 = 0.3, M3 = 05

Defective: M1: 0.05. M2 = 0.03, M3 = 0.01;

P(A3|B) = ?

P(A1) = 0.2, P(A2) = 0.3, P(A3) = 0.5

P(B|A1) = 0.05, P(B|A2) = 0.03, P(B|A3) = 0.01;

P(B) = summ(P(B|Ai) *P(Ai)) = 0.05*0.2 + 0.03 *0.3 + 0.01*0.5 = 0.024

P(A3|B) = P(A3) * P(B|A3) / P(B) = 0.5*0.01/0.024 = 5/24

24.4 EXERCISE PROBLEMS ON BAYES THEOREM

PDFs available in downloads;

24.5 NAIVE BAYES ALGORITHM

Naïve: Simplistic, Unsophisticated;

Probabilistic model [edit]

Abstractly, naive Bayes is a conditional probability model: given a problem instance to be classified, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$ representing some n features (independent variables), it assigns to this instance probabilities

$$p(C_k \mid x_1, \ldots, x_n)$$

for each of K possible outcomes or *classes* C_k.[7]

The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as

$$p(C_k \mid \mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$$

In plain English, using Bayesian probability terminology, the above equation can be written as

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on C and the values of the features x_i are given, so that the denominator is effectively constant. The numerator is equivalent to the joint probability model

$$p(C_k, x_1, \ldots, x_n)$$

which can be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$$\begin{aligned} p(C_k, x_1, \ldots, x_n) &= p(x_1, \ldots, x_n, C_k) \\ &= p(x_1 \mid x_2, \ldots, x_n, C_k)\, p(x_2, \ldots, x_n, C_k) \\ &= p(x_1 \mid x_2, \ldots, x_n, C_k)\, p(x_2 \mid x_3, \ldots, x_n, C_k)\, p(x_3, \ldots, x_n, C_k) \\ &= \ldots \\ &= p(x_1 \mid x_2, \ldots, x_n, C_k)\, p(x_2 \mid x_3, \ldots, x_n, C_k) \ldots p(x_{n-1} \mid x_n, C_k)\, p(x_n \mid C_k)\, p(C_k) \end{aligned}$$

Now the "naive" conditional independence assumptions come into play: assume that each feature x_i is conditionally independent of every other feature x_j for $j \neq i$, given the category C.

$$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k)\,.$$

Thus, the joint model can be expressed as

$$\begin{aligned} p(C_k \mid x_1, \ldots, x_n) &\propto p(C_k, x_1, \ldots, x_n) \\ &\propto p(C_k)\, p(x_1 \mid C_k)\, p(x_2 \mid C_k)\, p(x_3 \mid C_k) \cdots \\ &\propto p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)\,. \end{aligned}$$

Naïve: Features are conditionally independent of each other

24.6 TOY EXAMPLE: TRAIN AND TEST STAGES

	Predictors				Response
	Outlook	Temperature	Humidity	Wind	Class Play=Yes Play=No
Day1	Sunny	Hot	High	Weak	No
Day2	Sunny	Hot	High	Strong	No
Day3	Overcast	Hot	High	Weak	Yes
Day4	Rain	Mild	High	Weak	Yes
Day5	Rain	Cool	Normal	Weak	Yes
Day6	Rain	Cool	Normal	Strong	No
Day7	Overcast	Cool	Normal	Strong	Yes
Day8	Sunny	Mild	High	Weak	No
Day9	Sunny	Cool	Normal	Weak	Yes
Day10	Rain	Mild	Normal	Weak	Yes
Day11	Sunny	Mild	Normal	Strong	Yes
Day12	Overcast	Mild	High	Strong	Yes
Day13	Overcast	Hot	Normal	Weak	Yes
Day14	Rain	Mild	High	Strong	No

To determine: P(play = yes $|x_q$) and P(play = No$|x_q$);

P(Outlook=o|Class $_{Play=Yes\|No}$)	Frequency		Probability in Class	
Outlook =	Play=Yes	Play=No	Play=Yes	Play=No
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rain	3	2	3/9	2/5
	total= 9	total=5		

P(Temperature=t|Class $_{Play=Yes\|No}$)	Frequency		Probability in Class	
Temperature =	Play=Yes	Play=No	Play=Yes	Play=No
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
	total= 9	total=5		

Building counts and calculating probability priors, likelihood and evidence;

In the training phase of Naïve Bayes we calculate all likelihood probabilities and evidence probability;

Time complexity: O(ndc) through optimization O(n)

Space complexity: O(dc) d features and c classes

$$P(Class_{Play=Yes}|x') = [P(Sunny|Class_{Play=Yes}) \times P(Cool|Class_{Play=Yes}) \times P(High|Class_{Play=Yes}) \times P(Strong|Class_{Play=Yes})] \times P(Class_{Play=Yes})$$

$$= 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.0053$$

$$P(Class_{Play=No}|x') = [P(Sunny|Class_{Play=No}) \times P(Cool|Class_{Play=No}) \times P(High|Class_{Play=No}) \times P(Strong|Class_{Play=No})] \times P(Class_{Play=No})$$

$$= 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.0205$$

These probabilities are proportional;

P(class = No|x') > P(class = Yes|x'); thus the prediction will be class = No for x1 data point;

Test Time complexity: O(dc)

kNN space complexity: O(nd)

As compared to kNN Naïve Bayes is space efficient at run time; we can have low latency applications;

Refer: shatterline, Naïve Bayes

24.7 NAIVE BAYES ON TEXT DATA

Naïve Bayes is applied successfully to text data;

Task: Given text, compute probability of the text belonging to a class;

Text is vectorized;

P(y=1|text) Proportional to P(y=1|W1, W2, ….., Wd);

Proportional to P(y=1) * P(W1|y=1) * ….. P(Wd|y=1)

P(y = 1) is class prior and P(W1|y) is likelihood;

Naïve Bayes is often used as baseline benchmark model for text classification and suitable problems; all other algorithms are compared with Naïve Bayes performance;

24.8 LAPLACE/ADDITIVE SMOOTHING

At the end of training all the likelihoods and priors are computed;

At test time:

Say we find a new word for which likelihood is not available;

We will use Laplace smoothing (not Laplacian smoothing) as the word is not present in training data ideas such as making its likelihood as 0 or 1 or 0.5 is not right; thus we will add a smoothing value to the numerator and denominator for likelihood probability of the new word;

$P(W' | y=1) = (0 + \alpha)/(n1 + \alpha k)$

N1 = # of data points for y = 1

Smoothing coefficient, α != 0; generally = 1

k = number of unique values W' can take (for a categorical feature it can k = # of unique categories)

When α = large; the likelihood will be around 0.5;

Laplacian smoothing is applied to all words in training data and also to new words that occur in test data;

We have an additive smoothing and generally 1 additive smoothing is applied that is Laplace smoothing with $\alpha = 1$;

24.9 LOG-PROBABILITIES FOR NUMERICAL STABILITY

We have probability values which are bound between 0 and 1;

While the probabilities are multiplied the result will be extremely low which will affect our model performance and result interpretation; we will also land in rounding up errors for low decimal values;

We will take logarithm of probabilities to avoid the problems of multiplying values between 0 and 1;

24.10 BIAS AND VARIANCE TRADEOFF

Naïve Bayes: High Bias → Underfitting and High Variance → Ovefitting;

We have α as hyper parameter;

When α = 0 → small change in Dtrain results in large change in the model; high variance, overfitting;

When α = Very large; all likelihoods will be equal to 0.5; this results in underfitting, a high bias model; this will result in predicting majority class as class labels for all test data points;

In kNN k is determined by using cross validation, similarly α is determined by cross validation;

24.11 FEATURE IMPORTANCE AND INTERPRETABILITY

We have likelihood probabilities for all features;

Sort the probability values in descending order; for each class we will get an order of features which are important;

Features which have high likelihoods are most important features in classifying a data point;

Interpretability: Based on likelihood probability of features we can get why a data point is classified to a certain class;

24.12 IMBALANCED DATA

Class priors favors dominating class while comparing probabilities of a data point features belong to a class;

We can use upsampling or downsampling or drop prior probabilities;

Naïve Bayes is modified to account for class imbalance (less used)

When laplace smoothing is applied: alpha impacts more for minority class;

We can have different α values for different classes;

24.13 OUTLIERS

Outliers at test time are taken care by Laplace smoothing; and if a word occurs less frequently then discard that word;

24.14 MISSING VALUES

No case of missing value for text data;

Categorical data: Nan can be considered as another category

Numerical: Imputation

24.15 HANDLING NUMERICAL FEATURES (GAUSSIAN NB)

Naïve Bayes developed for numerical features as Gaussian Naïve Bayes;

Let us assume that the numerical feature follows a Gaussian distribution with some mean and standard deviation; before this we need to consider the data points that belong to the class in consideration; after considering the **class data points** only we can compute the probability from PDF of the distribution of the feature; the distribution PDF can be computed from **mean, standard deviation** and the assumption of **Gaussian distribution**;

Naïve Bayes on numerical features with assumption of Gaussian is called as Gaussian Naïve Bayes; on binary feature we will have Bernoulli Naïve Bayes; we can also have Multinomial Naïve Bayes when the distribution is assumed to be Multinomial;

Naïve Bayes has a fundamental assumption that all features are conditionally independent;

24.16 MULTICLASS CLASSIFICATION

Inherent property of Naïve Bayes; can be extended to multi classes;

We can directly compute class based probabilities;

24.17 SIMILARITY OR DISTANCE MATRIX

kNN can be implemented for an easily distance matrix;

Naïve Bayes cannot be directly applied on distance matrix; we need probability values;

24.18 LARGE DIMENSIONALITY

Naïve Bayes can be extensively used for text classification; as dimensionality increases we need to consider using log of probabilities;

24.19 BEST AND WORST CASES

If conditional independence deters NB performance deteriorates;

NB can still work reasonably well; as opposed to theoretical rigor where NB should work only for conditionally independent features;

NB is extensively used for Text classification and categorical features; NB is not much used to real valued features as real world distributions come in varied forms other than Gaussian;

NB is interpretable, we have feature importance, low runtime complexity, low train time complexity, low run time space complexity; NB is basically performing counting;

NB can easily overfit and is tackled using Laplace smoothing;

24.20 CODE EXAMPLE

Sklearn Naïve Bayes module;

Different ML algorithms are implemented by just changing a single line of code;

In ML the task is to understand the applications of different algorithms;

24.21 REVISION QUESTIONS

1. What is Conditional probability?
2. Define Independent vs mutually exclusive events?
3. Explain Bayes Theorem with example?
4. How to apply Naive Bayes on Text data?
5. What is Laplace/Additive Smoothing?
6. Explain Log-probabilities for numerical stability?
7. In Naive Bayes how to handle Bias and Variance tradeoff?
8. What Imbalanced data?
9. What is Outliers and how to handle outliers?
10. How to handle Missing values?
11. How to Handling Numerical features (Gaussian NB)
12. Define Multiclass classification?

Chapter 25: LOGISTIC REGRESSION

25.1 GEOMETRIC INTUITION OF LOGISTIC REGRESSION

Logistic Regression is actually a classification technique:

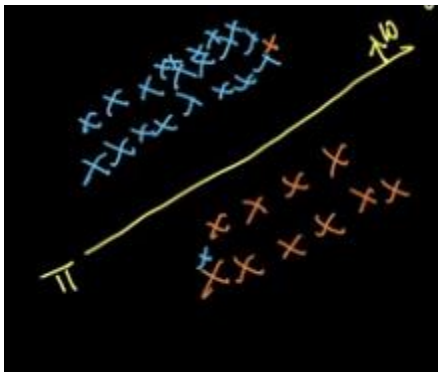
Logistic Regression can be interpreted in terms of Geometry, Probability and loss function;

If data is linearly separable (a hyper plane can separate the data points into two classes);

Equation of plane: $W^T x + b = 0$

If hyper planes passes through origin then $b = 0$; $W^T x = 0$

Logistic Regression: assumption: Classes are almost or perfectly linearly separable;



Task: Find plane that separates the data points;

Assumptions:

kNN: Neighborhood similarity

NB: Conditional Independence

Log Reg: Linearly separable

Negative data points are labeled -1 instead of 0;

Distance of a point from the hyper plane is: $d_i = W^T x_i / ||W||$

If W is a unit normal vector to the hyper plane:

$||W|| = 1$

Then, $d_i = W^T x_i$;

Now we will have a classifier where if the data point is in the same direction of the normal vector then it belongs to positive class else negative;

For this we have: $W^T x_i > 0$ then $y_i = +1$

For positive data points: $y_i W^T x_i > 0$ → the data point is correctly classified

y_i = true label = +1 and $W^T x_i > 0$

For negative data points: $y_i W^T x_i > 0$ → the data point is correctly classified

y_i = true label = -1 and $W^T x_i < 0$

For all data points: $y_i W^T x_i > 0$ → the data point is correctly classified

If $y_i W^T x_i < 0$ → the data point is incorrectly classified;

The ML model should have minimum number of misclassifications;

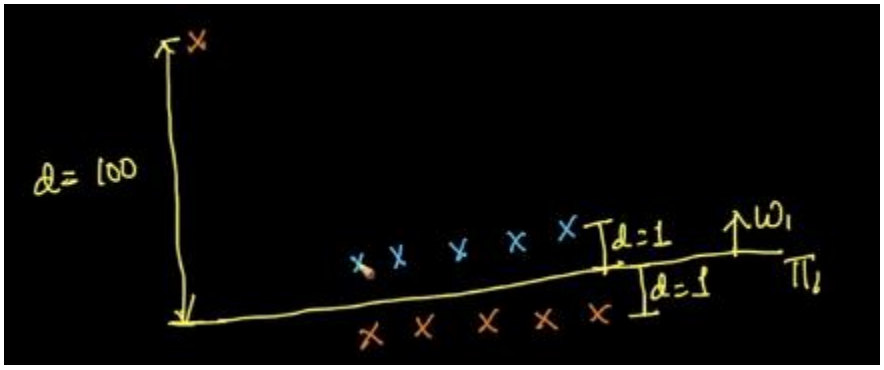
Thus find W that maximizes summ($y_i W^T x_i$)

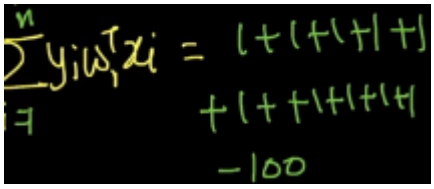
$W^* = \text{argmax}(\text{summ}(y_i W^T x_i))$

25.2 SIGMOID FUNCTION: SQUASHING

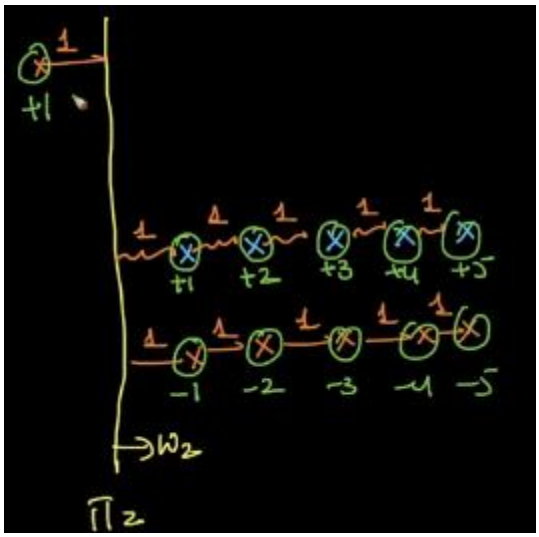
($y_i W^T x_i$) = signed distance;

Outliers impact W values largely;





Case 2:



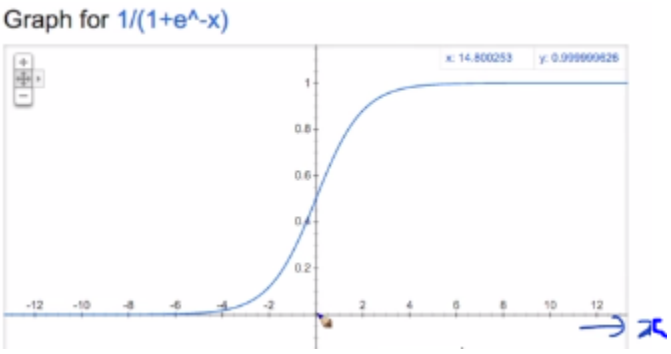
This formulation is largely impacted by outliers or any changes in the training data;

We will use squashing to reduce the effect of this;

Idea of squashing: if the signed distance is small use it as it is and if the signed distance is large make it small;

We have sigmoid function which has this property; we can have other functions

Sigmoid(x): $1/(1+e^{-yWx})$



So if a point lies on the Hyperplane we will have $W^T x = 0$; then this point belonging to positive or negative class is 0.5; this can be seen from sigmoid(0) = 0.5;

Max sum of signed distances was outlier prone;

Maximize sum of sigmoid of signed distances which is outlier resistant;

$W^* = \text{argmax summ } 1/(1+ \exp(-y_i W^T x_i)$

The distances are squashed from [– infinity, + infinity] to [0, 1]

Sigmoid is easily differentiable and has probabilistic interpretation which helps in solving the optimization problem;

25.3 MATHEMATICAL FORMULATION OF OBJECTIVE FUNCTION

Sigmoid function is a monotonic function;

A function is monotonic which increases with x at all values;

Log(x) is a monotonic function;

If a function is monotonic then when applied a monotonic function retains same maxima or minima;

Argmin f(x) = argmin g(f(x)) if g(x) is monotonic;

$W^* = \text{argmax summ } 1/(1+ \exp(-y_i W^T x_i)$

$= \text{argmax summ } \log(1/(1+ \exp(-y_i W^T x_i))$

$= \text{argmax summ } -\log((1+ \exp(-y_i W^T x_i))$

$= \text{argmin summ } \log((1+ \exp(-y_i W^T x_i))$

$= \text{argmin summ } \log((\exp(-y_i W^T x_i))$

$= \text{argmin summ } (-y_i W^T x_i)$

$= \text{argmax summ } (y_i W^T x_i)$

$y_i = \{-1, +1\}$

The formulation is same but the sigmoid version will be less impacted by outliers

Probabilistic interpretation:

$W^* = \text{argmin summ}(-y_i \log p_i - (1 - y_i)\log(1 - p_i))$; $p_i = \text{sigmoid}(W^T x)$

$y_i = \{0, 1\}$

Above Geometric interpretation:

$W^* = \text{argmin summ } \log((1+ \exp(-y_i W^T x_i))$

$y_i = \{-1, +1\}$

25.4 WEIGHT VECTOR

$W^* = \text{argmin summ } \log((1+ \exp(-y_i W^T x_i))$

W* = Weight vector = d dimensional vector;

Geometric intuition: The W vector is the optimal weight vector normal to a hyper plane that separates data points into classes where positive data points are in the direction of W;

Decision: Given x_q predict y_q

For Logistic Regression: If $W^T x_q > 0$ then $y_q = +1$; If $W^T x_q < 0$ then $y_q = -1$, if the point is on the hyper plane we cannot determine the class of the query point;

Probabilistic interpretation: $P(y_q = +1) = \text{sigmoid } (W^T x_q)$

(Remember the need for sigmoid function: squashing)

Interpretation of Weight vectors:

Case 1: If W_i = +ve then if x_{qi} increases $W_i x_{qi}$ increases, sigmoid $(W^T x_q)$ increases, then $P(y_q=+1)$ increases;

Case2: If W_i = -ve then if x_{qi} increases $W_i x_{qi}$ decreases, sigmoid ($W^T x_q$) decreases, then $P(y_q=+1)$ decreases and $P(y_q=-1)$ increases;

25.5 L2 REGULARIZATION: OVERFITTING AND UNDERFITTING

$W^* =$ argmin summ $\log((1+ \exp(-y_i W^T x_i))$

Let $Z_i = y_i W^T x_i / ||W||$

Then W^* = argmin summ(i = 1 to n) ($\log(1+\exp(-Z_i))$)

$\exp(-Z_i)) >= 0$; $1 + \exp(-Z_i)) >= 1$; $\log(1+\exp(-Z_i))) >= 0$;

Minimum value of summ(i = 1 to n) ($\log(1+\exp(-Z_i))$ is 0;

If Z_i tends to + infinity then the summ(i = 1 to n) ($\log(1+\exp(-Z_i))$ is 0;

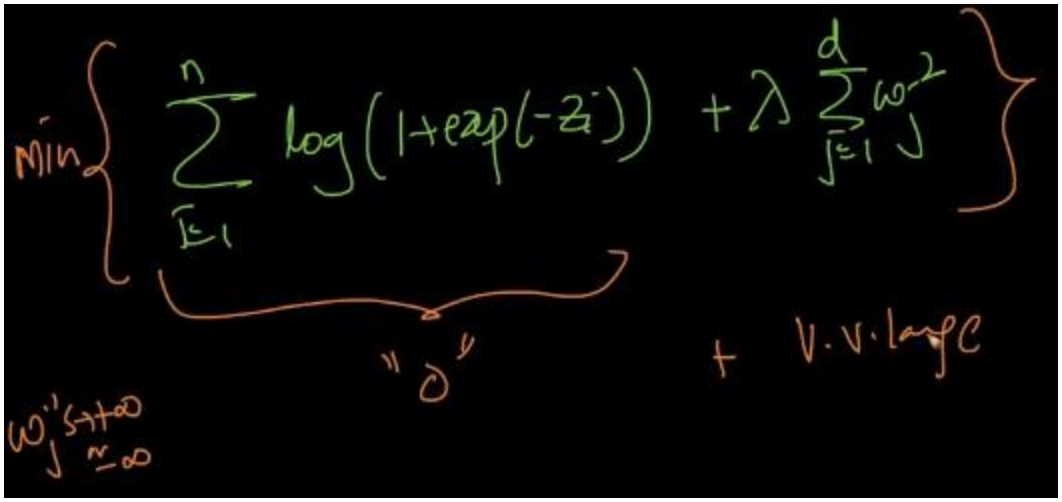
If the selected W results in correctly classifying all training points and if Z_i tends to infinity, then W is the best W on training data; this is a case of overfitting on training data as this does not guarantee good performance on test data; the train data may contain outliers to which the model has been perfectly fitted;

The overfitting tendency is constrained by using regularization;

W^* = argmin summ $\log((1+ \exp(-y_i W^T x_i)) + \lambda W^T W$

= argmin summ $\log((1+ \exp(-y_i W^T x_i)) + \lambda^* ||W||_2$

The regularization term will constrain W from reaching +infinity or - infinity;



λ is a hyper parameter of regularization; this is determined using cross validation;

If λ = 0 there is not regularization; the loss term optimization results in an overfitting model → high variance;

If λ is large then the loss term is diminished; the training data does not participate in the optimization and we are just optimizing for the regularization term; this leads to an underfitting model → high bias;

25.6 L1 REGULARIZATION AND SPARSITY

Optimization problem: $W^* = \text{argmin summ } \log((1+ \exp(-y_i W^T x_i)) + \lambda W^T W$

$=$ logistic loss + Regularization

Objective for Regularization: Avoid W reaching infinity;

L1 Regularization:

$W^* = \text{argmin logistic loss for training data} + \lambda||W||_1$

L1 Regularization induces Sparsity in the W vector;

Less important features become vanished in Logistic Regression with L1 Regularization;

If L2 Regularization is used Weight for less important features becomes small but remains non-zero;

Elastic net: $W^* = \text{argmin summ}(i = 1 \text{ to } n) \log(1+\exp(-Z_i) + \lambda 1||W||_1 + \lambda 2||W||_2$

25.7 PROBABILISTIC INTERPRETATION: GAUSSIAN NAIVE BAYES

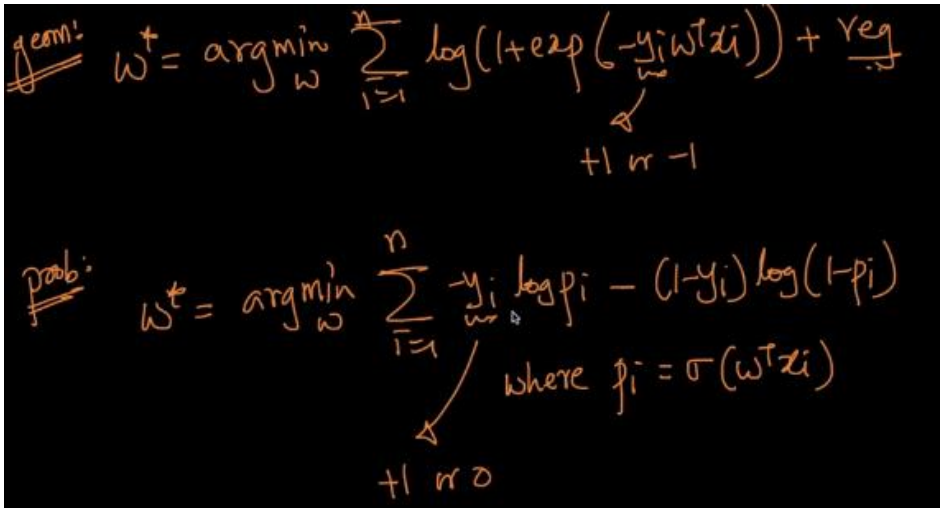
No book encountered that covers all of geometric, probabilistic and loss minimization interpretations of logistic regression;

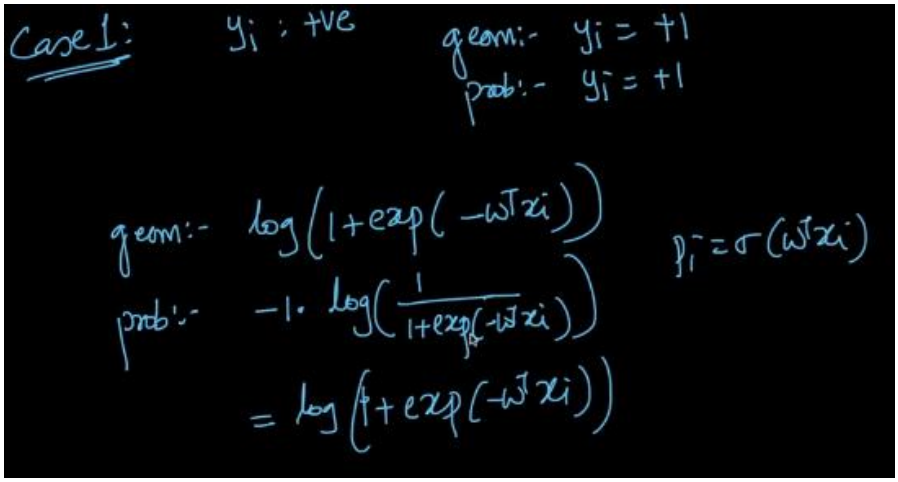
Naive Bayes: real valued features are Gaussian distributed and class label is a Bernoulli random variable;

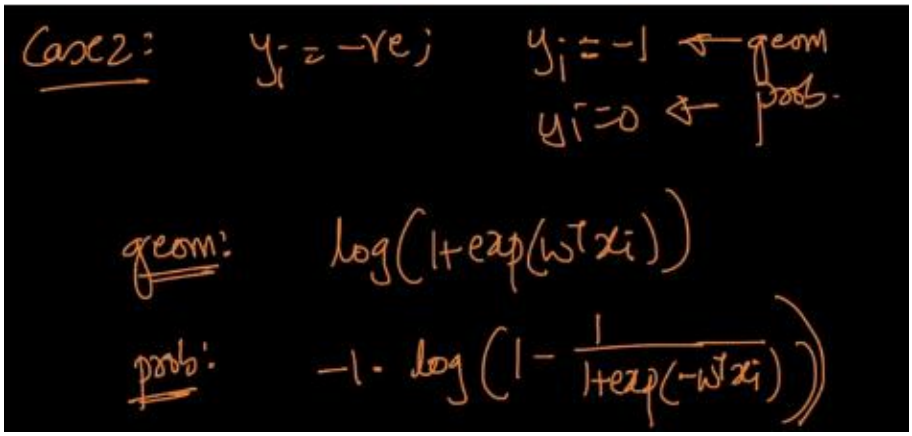
Assumption: X_i is a feature, $P(X_i|y_i)$ is a Gaussian distribution and X_i's are conditionally independent

$P(Y=1|X) = 1/(1+\exp(W^T x))$

Logistic Regression: Gaussian Naïve Bayes + Bernoulli;





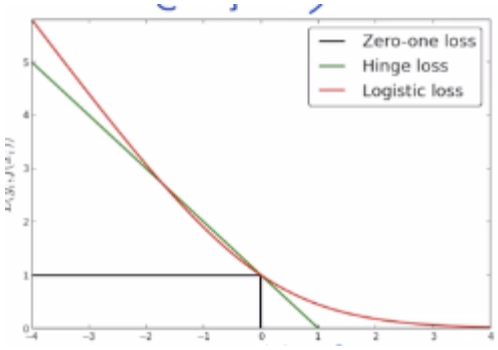


Link: https://www.cs.cmu.edu/~tom/mlbook/NBayesLogReg.pdf

25.8 LOSS MINIMIZATION INTERPRETATION

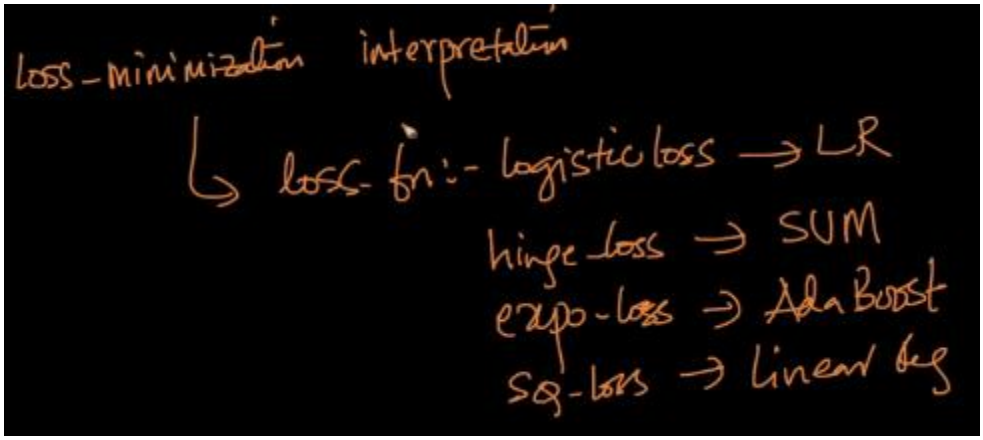
W* = argmin number of misclassifications;

For optimization the function should be differentiable;

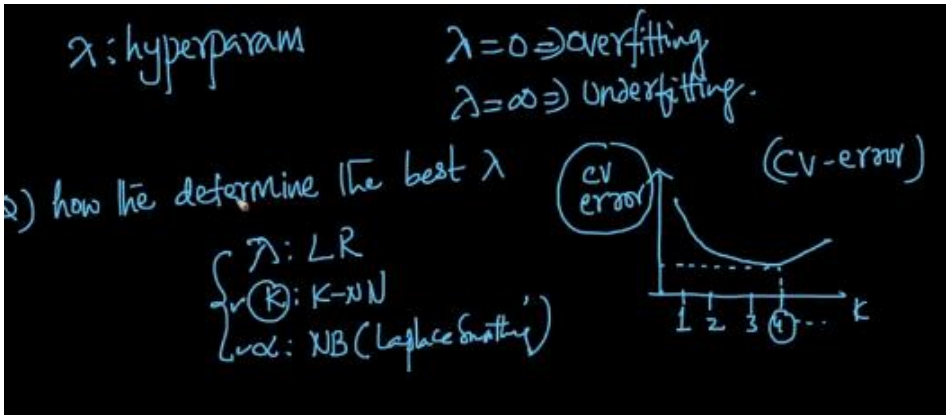


Logistic loss is a good approximation of 0-1 loss (step loss)

With hinge loss we will have Support Vector Machines;



25.9 HYPERPARAMETER SEARCH: GRID SEARCH AND RANDOM SEARCH



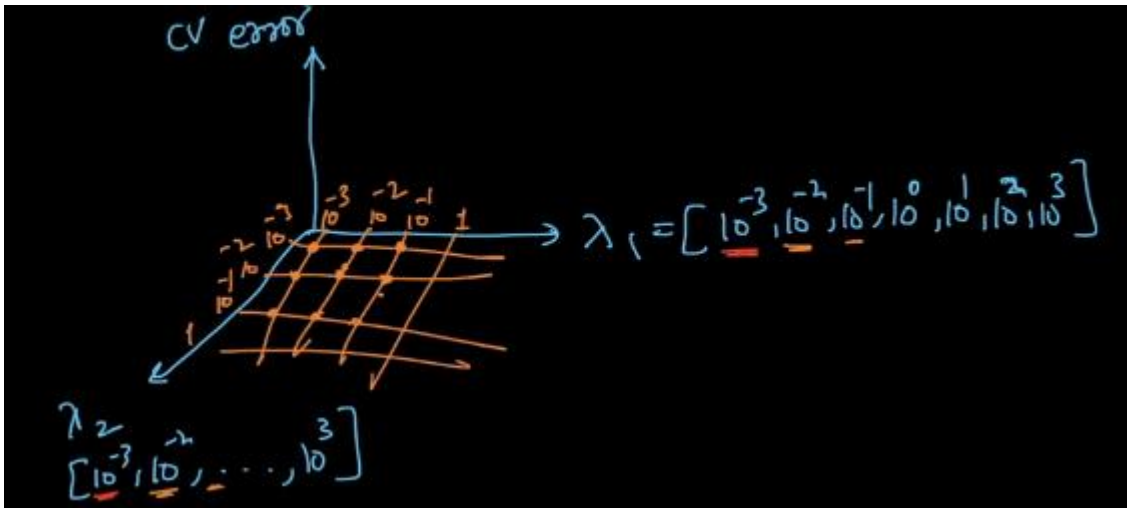
K in kNN is an integer; Logistic Regression λ is a real valued hyper parameter;

Hyper parameter tuning technique: Grid Seach (Brute force)

For λ in [0.001, 0.01, 0.1, 1, 10, 100]:

> Compute cross validation error and select the best hyper parameter value from the plot;

Elastic Net: λ1 and λ2;



Multiple computations or training is required;

As # of hyper parameters increase, the # of training instances increase exponentially;

Random Search: avoids brute force and reduces the time spent for hyper parameter tuning by looking at a smaller set of hyper parameter choices; the technique considers random hyper parameter values for computing cross validation error in turn provides best hyper parameter choices that optimizes the algorithm;

25.10 COLUMN STANDARDIZATION

Mean centering and scaling; used in kNN;

In logistic regression it is also mandatory to perform column standardization as we are using distances to compute the weight vector and the scale of features impacts the distance values;

25.11 FEATURE IMPORTANCE AND MODEL INTERPRETABILITY

We have d features and through optimization we determine W vector of d dimensionality, each element in W vector corresponds to respective features;

If all features are independent:

Weight element values indicate how important the feature is;

In kNN: we have forward feature selection;

Forward feature selection is algorithm independent can be applied for NB and LR

In NB: Probability values give feature importance (Conditional probabilities)

In Logistic Regression: Weights can be used to determine feature importance;

If absolute value of weight is large then contribution of the corresponding feature is large, thus this feature is important;

25.12 COLLINEARITY OF FEATURES

Feature importance interpretation is done from weight vectors assuming independence;

If there is co-linearity then we cannot interpret feature importance from weight vector;

Two features are collinear if a feature can be expressed as a function of other feature;

Multi collinear feature is a feature that can be expressed as a function of multiple features;

Weight vectors are impacted by multi collinear features;

To use Weight vector for interpreting feature importance then we need to remove multi collinear features;

Multi collinear feature can be determined by adding noise to features; if we add small noise to the values that is perturbation to the feature and if the weight vector varies (after training)a lot then the features are multi collinear; we then cannot use W vector for feature importance;

Multi collinear test is mandatory;

25.13 TRAIN & RUN TIME SPACE & TIME COMPLEXITY

Training Logistic Regression: Solving the optimization problem using Stochastic Gradient Descent;

Train time:

Time complexity: O(nd)

Run time:

Space: O(d)

Time: O(d)

Logistic Regression is applicable for low latency applications; popular algorithm for internet companies, memory efficient at run time;

If dimensionality is large, L1 regularization inducing Sparsity can be applied to reduce run time complexity; As the hyper parameter increases Sparsity also increases; this achieves both a suitable bias variance trade off and the low latency requirements;

As λ increases, bias increases, latency decreases and Sparsity increases; but the model is just a working model not an optimized model as regularization and Sparsity are induced;

25.14 REAL WORLD CASES

Decision surface: Linear Hyperplane

Assumption: data is linearly separable or almost linearly separable;

We also have good interpretability;

Imbalanced data: Upsampling or down sampling; with imbalanced data we will have incorrect hyper planes that can optimize the loss function where the hyper plane can make all the data points lie on one side optimizing the loss function for majority class;

Outliers: due to sigmoid function the impact is less;

We can compute W from Dtrain through training and generate distances through $W^T x_i$; the points that have large distances are outliers can be removed to make Dtrain free from outliers; the model is retrained on the outlier free dataset to get final Weight vector

Mssing values: mean, median or model imputation;

Multi class classification: One Vs Rest and Max entropy/Softmax/Multinomial LR models

Input similarity matrix: Extension of LR, Kernel LR (SVM) can be trained on similarity matrix;

Best case:

> Data is linearly separable, low latency, faster to train; high dimensionality (higher chance that data is linearly separable and we can use L1 regularization)

Worst case:

> Non linear data;

25.15 NON-LINEARLY SEPARABLE DATA & FEATURE ENGINEERING

We need to transform features to make it linearly separable;

A circularly separable data set can be transformed into linearly separable data set by transforming features into a quadratic space;

$F1^2 + F2^2 = R^2$ is not linear in F1 and F2 but linear in $F1^2$ and $F2^2$;

Most important aspects of applied AI is:

1. Data analysis and Visualization
2. Feature Engineering (the difference between candidates, Deep Learning almost automates feature engineering)
3. Bias variance trade off

We can use multiplication of features, polynomials, trigonometric Boolean, exponential or logarithmic transformations for making data linearly separable and even function combinations of several features;

25.16 CODE SAMPLE: LOGISTIC REGRESSION, GRIDSEARCHCV, RANDOMSEARCHCV

Load libraries and datasets

C increases, overfitting happens;

Sklearn.linear_model.LogisticRegression; sklearn.model_selection.GridSearchCV

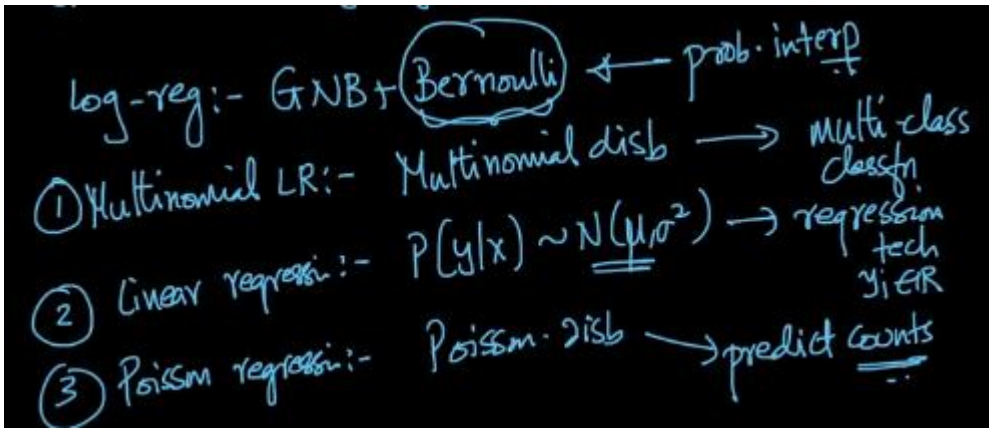
```
#code source: http://occam.olin.edu/sites/default/files/DataScienceMaterial
from sklearn.model_selection import train_test_split
from sklearn.grid_search import GridSearchCV
from sklearn.datasets import *
from sklearn.linear_model import LogisticRegression

data = load_breast_cancer() #refer: http://scikit-learn.org/stable/modules/

tuned_parameters = [{'C': [10**-4, 10**-2, 10**0, 10**2, 10**4]}]
X_train, X_test, y_train, y_test = train_test_split(data.data, data.target,


#Using GridSearchCV
model = GridSearchCV(LogisticRegression(), tuned_parameters, scoring = 'f1'
model.fit(X_train, y_train)

print(model.best_estimator_)
print(model.score(X_test, y_test))
```

GridSearchCV will train for 5 hyper parameter values;

Experiment to understand different behavior of the models;

As hyper parameter value changes we can have varying weights;

25.17 EXTENSIONS TO LOGISTIC REGRESSION: GENERALIZED LINEAR MODELS (GLM)



Link: http://cs229.stanford.edu/notes/cs229-notes1.pdf

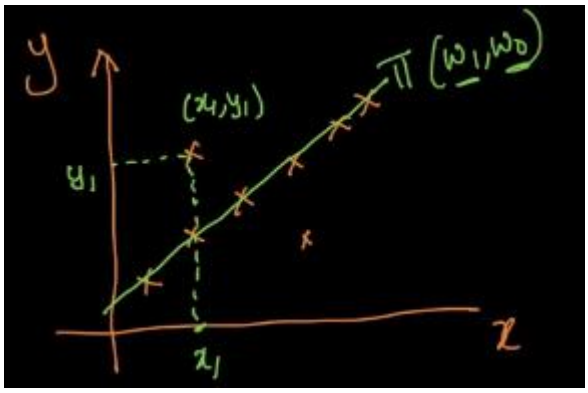
Chapter 26: LINEAR REGRESSION

26.1 GEOMETRIC INTUITION OF LINEAR REGRESSION

Logistic Regression: A two class classification technique

Linear Regression: Find a hyper plane that best fits the training data (continuous variable data)

$$y_i = W^T x_i + b$$

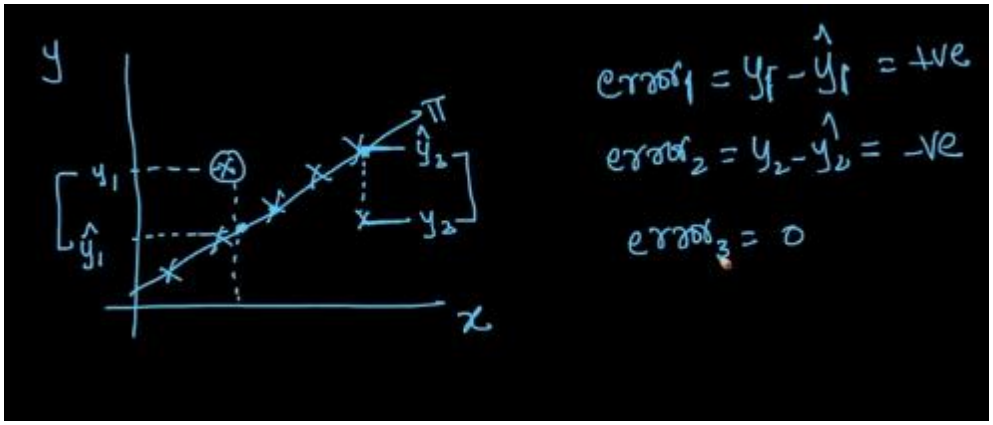


π: best fit plane; the points that are not on the hyper plane have errors due to incorrect prediction;

error = true value – predicted value

Task: Minimize the sum of errors across the training data;

26.2 MATHEMATICAL FORMULATION



As there are positive and negative valued errors we need to make the values free from signs; we can use squared error;

Formulation:

$$(W^*, W_0) = \text{argmin summ } (i = 1 \text{ to } n)\ (y_i - y_{pred})^2$$

$$= \text{argmin summ } (i = 1 \text{ to } n)\ (y_i - (W^T x_i + W_0))^2$$

Regularization:

$$(W^*, W_0) = \text{argmin summ } (i = 1 \text{ to } n)\ (y_i - (W^T x_i + W_0))^2 + \lambda ||W||_2$$

We can also use L1 regularization;

Loss minimization:

For classification: $Z_i = y_i f(x_i)$ for logistic regression we have sigmoid function, we can also have step function or hinge loss which form other ML algorithms;

For regression: $Z_i = y_i - f(x_i)$; $f(x_i) = W^T x_i + W_0$; and the loss function is squared loss;

Terminology:

1. Linear regression with L2 regularization is also referred to as "Ridge Regression"
2. Linear regression with L1 regularization is also referred to as "Lasso Regression".
3. Linear regression with l1+L2 regularization is also referred to as "Elastic Net Regression".

Regularization in Linear Regression:

For logistic regression we introduced regularization to constrain weight elements reaching infinity;

In real world we round up the values to an easily understandable decimal; this results in small errors even though we know the underlying relationships;

Without regularization we will have noise propagated to the weight values; to reduce the effect of noise in features on the weight vector we introduce regularization;

26.3 REAL WORLD CASES

No problem of imbalanced data;

Feature Importance and Interpretability: same as for Logistic Regression; we can use weight vector elements if the data does not have multi collinear features;

Outliers: In logistic regression we have sigmoid function that is squashing and limiting the impact of outliers;

In Linear Regression we will have squared loss; to remove outliers we can compute distances of a Hyperplane that best fitted on the training set; the data points that are very far from the hyper plane are removed and the hyper plane is regenerated to fit the outlier free data set; iterable up to satisfaction;

Outliers impact the model heavily; this technique of iterated removal of outliers from model training is called RANSAC;

26.4 CODE SAMPLE FOR LINEAR REGRESSION

Boston housing dataset; load data and split data; do EDA and Feature engineering;

Fit Linear Regression on the train data;

```
# code source:https://medium.com/@haydar_ai/learning-data-scien
from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm.fit(X_train, Y_train)

Y_pred = lm.predict(X_test)

plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
plt.show()
```

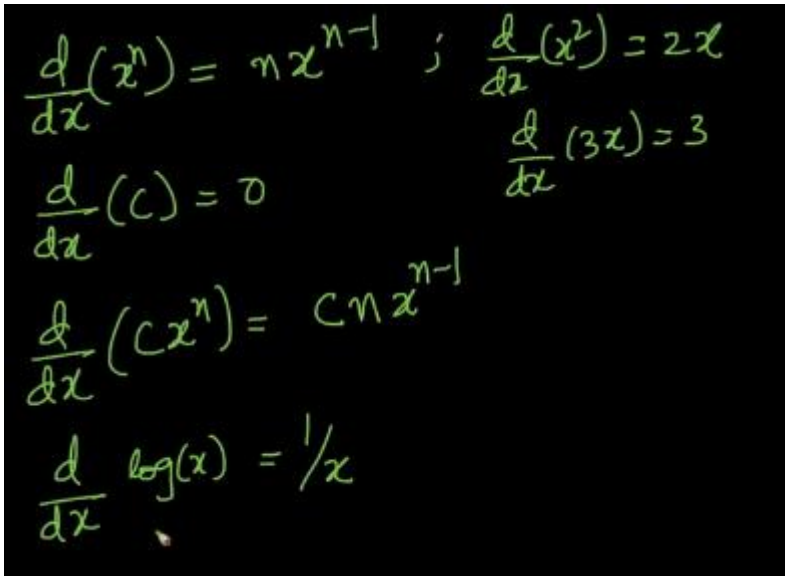
Chapter 27: SOLVING OPTIMIZATION PROBLEMS

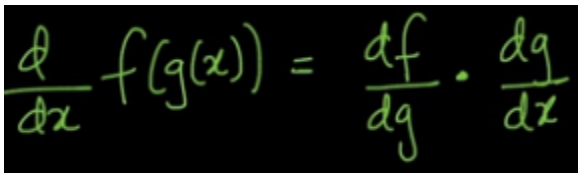
27.1 DIFFERENTIATION

y = f(x)

Differentiation of y with respect to x = dy/dx = df/dx = y' = f'

dy/dx = change in y due to change in x = (y2 – y1)/ (x2 – x1) = slope of the tangent to f(x)





27.2 ONLINE DIFFERENTIATION TOOLS

Link: https://www.derivative-calculator.net

27.3 MAXIMA AND MINIMA

Maxima: The maximum value a function takes;

Minima: The minimum value a function takes;

Slope at maxima / minima / saddle point = 0 = df/dx

F(x) = log(1 + exp(ax))

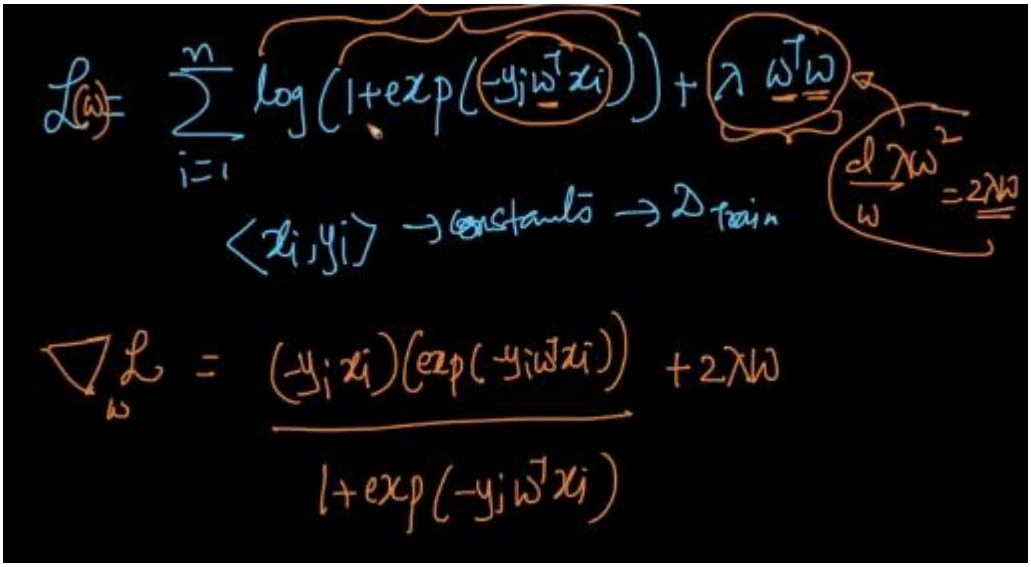
F’(x) = a exp(ax)/(1 + exp(ax))

Most of the functions cannot be readily solved; we will use SGD to solve optimization problems;

27.4 VECTOR CALCULUS: GRAD

Differentiation of vectors results in a vector; we will be applying element wise partial differentiation;

Logistic loss:



Solving the logistic loss is hard and thus we can use gradient descent technique

27.5 GRADIENT DESCENT: GEOMETRIC INTUITION

Iterative algorithm; initially we make a guess on the solution and we move towards the solution iteratively through solution correction;

Slope reaches zero when arriving at the optimum;

Gradient Descent:

1. Pick an initial point = x_{old} randomly
2. $X_{new} = x_{old} - r^* [df/dx]_{xold}$ (r = step size) with this gradient descent moves towards optimum;
3. Step 2 is update step which is iterated unless $x_{new} \sim x_{old}$;

4. In addition the gradient decreases at each iteration; first iteration update will be large, then successive updates decreases until optimum is reached; the update reduces as slope reduces

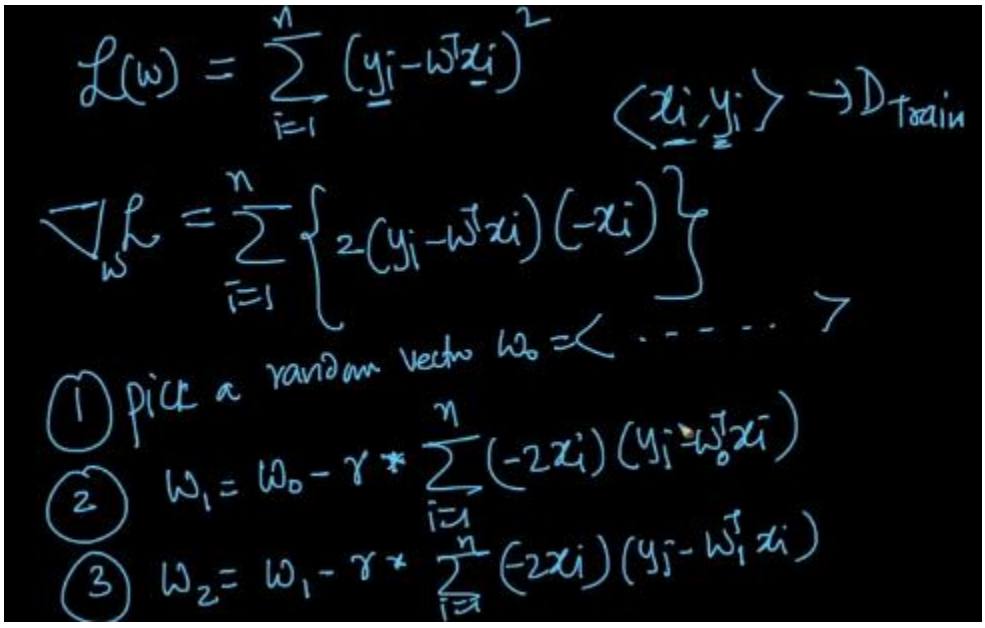
27.6 LEARNING RATE

Gradient Descent Update equation:

If learning rate does not reduce, gradient descent can jump over the optimum and this can be an iterative jump over where the algorithm does not reach the optimum; we are having oscillations without convergence;

We should reduce step size that is the learning rate is reduced at every iteration such that the convergence is guaranteed;

27.7 GRADIENT DESCENT FOR LINEAR REGRESSION



With Gradient descent we need to compute the updates over all data points which is expensive;

27.8 SGD ALGORITHM

Linear regression:

Gradient descent update:

$$W_{j+1} = W_j - r_s * \text{summ}(i = 1 \text{ to } n)(-2x_i)(y_i - W^T_j x_i)$$

$$W_{j+1} = W_j - r_s * \text{summ}(i = 1 \text{ to } k)(-2x_i)(y_i - W^T_j x_i)$$

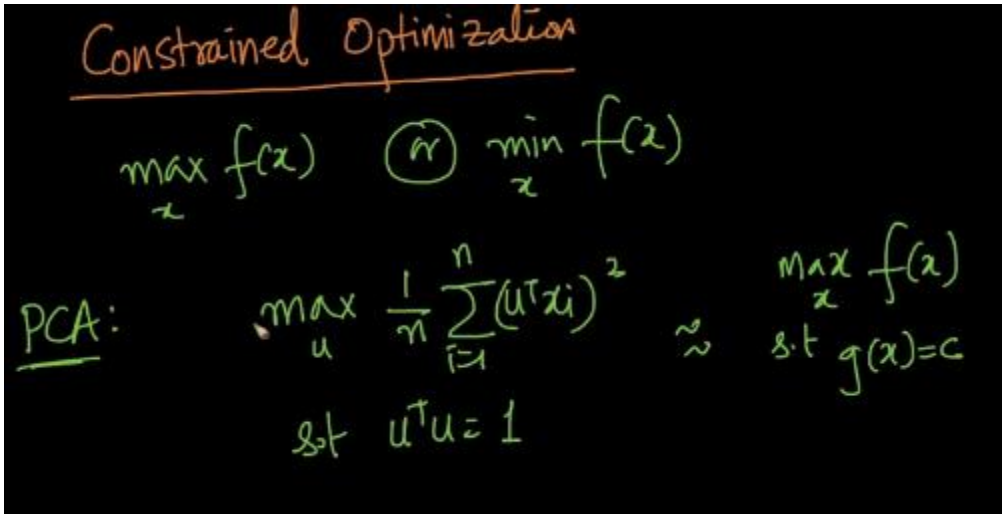
Changed iteration over n to k where we compute for k random points;

As we pick k random points we call this Gradient Descent as Stochastic Gradient Descent;

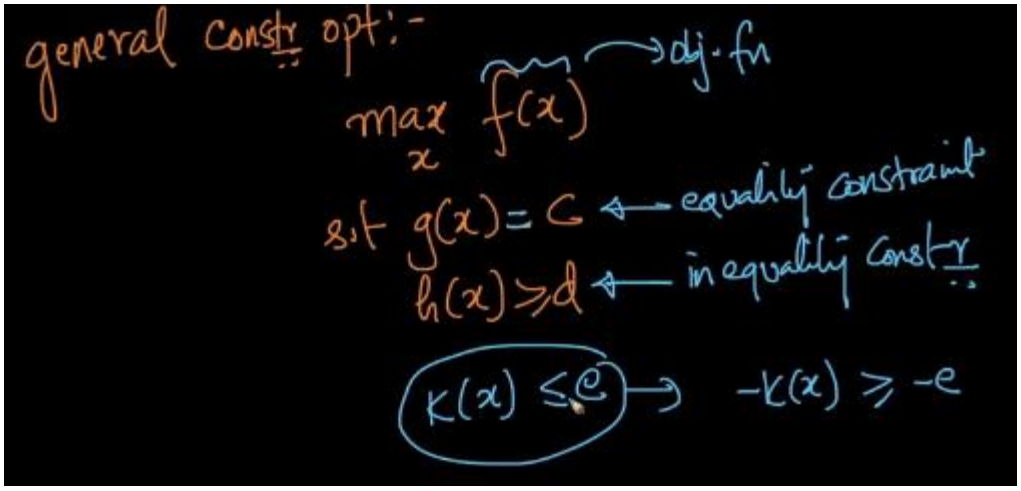
As k reduces from n to 1 the number of iterations required to reach optimum increases; At each iteration we have k and r changing; k is randomly picked between 1 and n which is called as batch size; r is reduced at each iteration progressively;

SGD is efficient; we are adding randomness to Gradient Descent to reduce time complexity at run time;

27.9 CONSTRAINED OPTIMIZATION & PCA

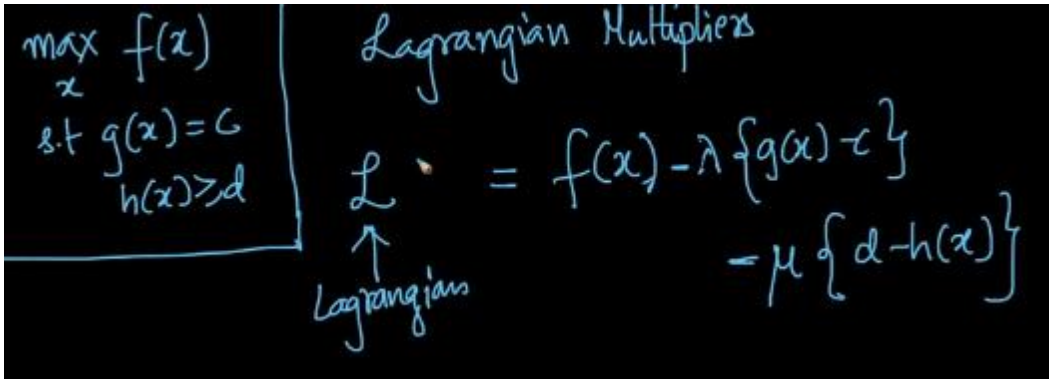


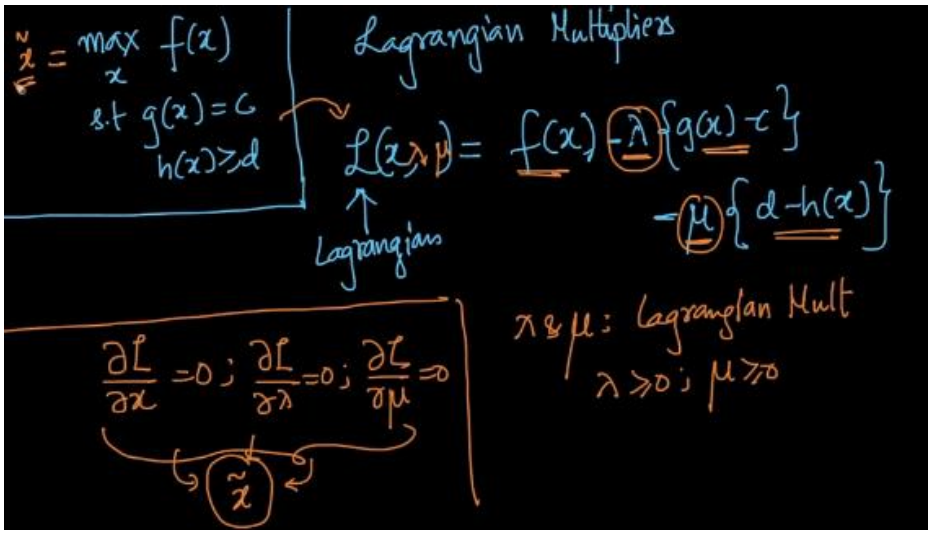
U^TU is constraint for the optimization objective function;



Finding optimum under constraints:

We need to modify the objective function using Lagrangian Multipliers;

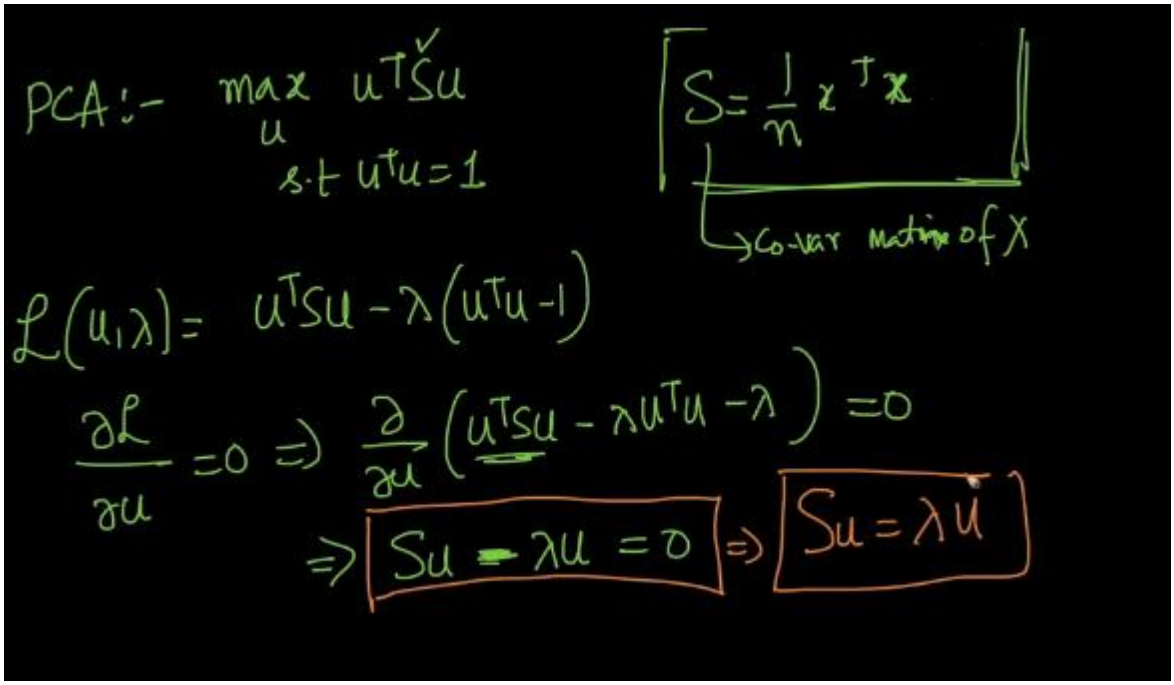




λ and μ are Lagrangian Multipliers which are >= 0

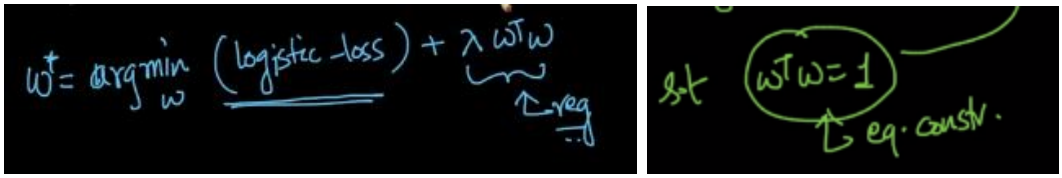
We can get solution for optimization by solving the partial derivatives of Lagrangian function with respect to x, λ and μ.

PCA can be reformulated as:



Su = λu, then λ = Eigen value and u is the Eigen vector of S;

27.10 LOGISTIC REGRESSION FORMULATION REVISITED



Lagrangian equivalent: L = logistic loss – $\lambda(1 - W^T W)$

Regularization is imposing an equality constraint on the logistic loss function;

Regularization in optimization formulation can be interpreted using equality constrained Lagrangian Multiplier;

27.11 WHY L1 REGULARIZATION CREATES SPARSITY?

L1 Regularization induces Sparsity into weight vector;

Sparsity implies that most of the elements of the weight vector are zero;

In optimization formulation for comparison Loss and λ can be ignored as they are same for both L1 and L2 regularizations;

L2 formulation has: min W for (W_1^2+W_2^2+ …. + W_d^2): this is a parabola;

L1 formulation has: min W for ($|W_1|$+$|W_2|$+ …. + $|W_d|$): this is a v shaped curve;

L2(W1) = $W1^2$; dL2/dW1 = 2*W1: derivative is a straight line

L1(W1) = |W1|; dL1/dW1 = +1 or -1: derivative is a step function

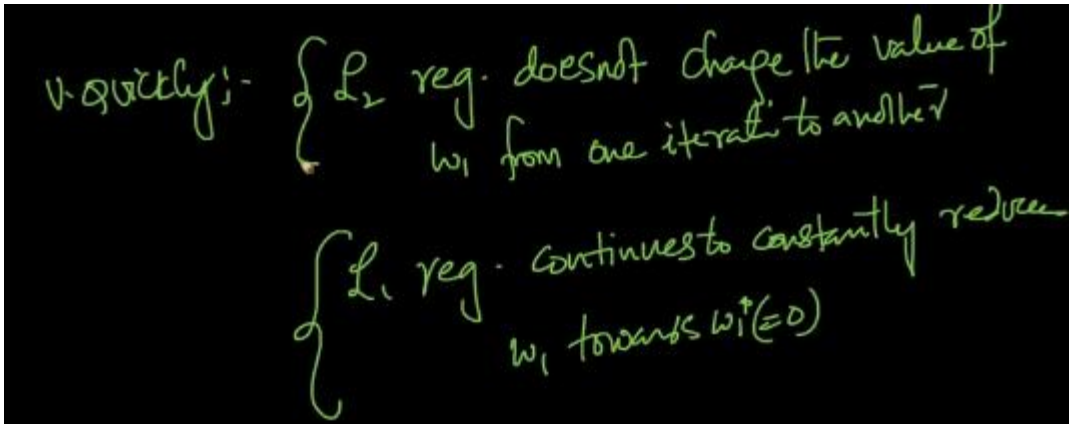
$W1_{j+1} = W1_j – r(dL2/dW1)_{W1j}$; for both L1 and L2

Let W1 is positive (as similarly we can do for W1 negative)

For L2: $W1_{j+1} = W1_j – r*(2*W1_j)$

For L1: $W1_{j+1} = W1_j – r*1$

L2 updates occurs less when compared to L1 updates as we reach closer to optimum; that is the rate of convergence decreases because in L2 regularization we have 2 * W1 *r which is less than r;



This happens because L1 derivative is constant and L2 derivative is not constant;

The chance of weights reaching 0 is more for L1 regularization as the derivative is constant and independent of the previous weight value; L2 regularization has derivatives reducing as the derivative is dependent on the previous iteration weight value which is converging to optimal;

27.12 REVISION QUESTIONS

1. Explain about Logistic regression?
2. What is Sigmoid function & Squashing?

3. Explain about Optimization problem in logistic regression.
4. Expalain Importance of Weight vector in logistic regression.
5. L2 Regularization: Overfitting and Underfitting
6. L1 regularization and sparsity.
7. What is Probabilistic Interpretation: Gaussian Naive Bayes?
8. Explain about Hyperparameter search: Grid Search and Random Search?
9. What is Column Standardization?
10. Explain about Collinearity of features?
11. Find Train & Run time space and time complexity of Logistic regression?

28.1 QUESTIONS & ANSWERS

1. After analysing the model, your manager has informed that your regression model is suffering from multicollinearity. How would you check if he's true? Without losing any information, can you still build a better model?(https://google-interview-hacks.blogspot.in/2017/04/after-analyzing-model-your-manager-has.html)
2. What are the basic assumptions to be made for linear regression?(https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1-2-copy-8/)
3. What is the difference between stochastic gradient descent (SGD) and gradient descent (GD)?(https://stats.stackexchange.com/questions/317675/gradient-descent-gd-vs-stochastic-gradient-descent-sgd)
4. When would you use GD over SDG, and vice-versa?(https://elitedatascience.com/machine-learning-interview-questions-answers)
5. How do you decide whether your linear regression model fits the data?(https://www.researchgate.net/post/What_statistical_test_is_required_to_assess_goodness_of_fit_of_a_linear_or_nonlinear_regression_equation)
6. Is it possible to perform logistic regression with Microsoft Excel?(https://www.youtube.com/watch?v=EKRjDurXau0)
7. When will you use classification over regression?(https://www.quora.com/When-will-you-use-classification-over-regression)
8. Why isn't Logistic Regression called Logistic Classification?(Refer :https://stats.stackexchange.com/questions/127042/why-isnt-logistic-regression-called-logistic-classification/127044)

External Resources:

1. https://www.analyticsvidhya.com/blog/2017/08/skilltest-logistic-regression/

2. https://www.listendata.com/2017/03/predictive-modeling-interview-questions.html

3. https://www.analyticsvidhya.com/blog/2017/07/30-questions-to-test-a-data-scientist-on-linear-regression/

4. https://www.analyticsvidhya.com/blog/2016/12/45-questions-to-test-a-data-scientist-on-regression-skill-test-regression-solution/

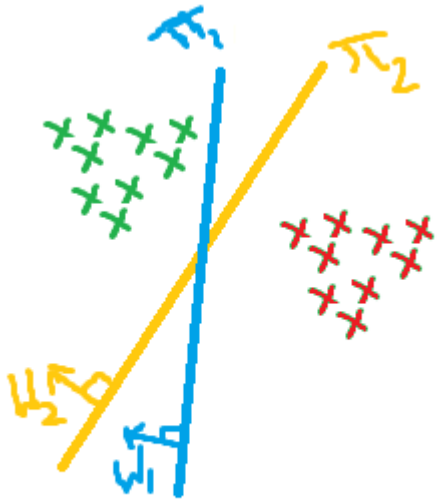
5. https://www.listendata.com/2018/03/regression-analysis.html

Chapter 29: Support Vector Machines (SVM)

29.1 Geometric Intuition

SVM – Popular Machine Learning Model – classification and regression problems

Let data points be as:



The data points can be separated with many hyper-planes. Say, with

In logistic regression, if a data point is very close to the hyper plane, then the probability of the point belonging to a class is very close to 0.5. For points away from hyper plane the probability will be close to 1.

The objective will be to separate the data points with a hyper plane which is as far as possible from the data points.

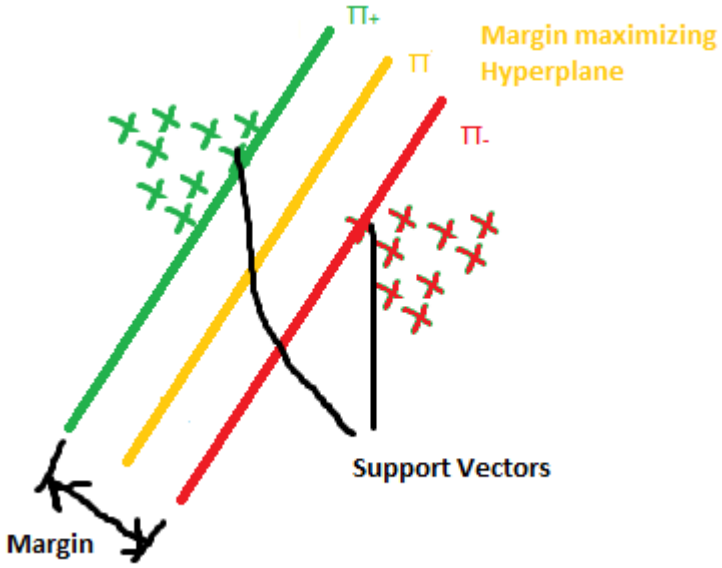
π_2 is better than π_1 as the hyper plane is as far as possible from all data points. Thus, π_2 can be taken as a margin maximizing hyper plane.

Take π_+ and π_- parallel to π_2 such that it touches first data point of the respective group. We get a margin and with SVM we maximize the width of this margin.

Objective: find π such that dist(π_+, π_-) (= margin) is maximum

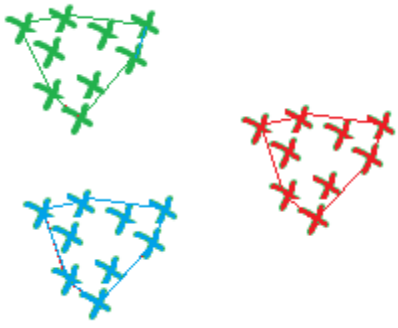
With this we will have lesser misclassifications and better generalization (s margin increases).

Support Vector: Say we have the margin maximizing hyper plane π, and we have π_+ and π_-, the data points through which these margin planes pass through are called support vectors. We have two support vectors in the following plot.



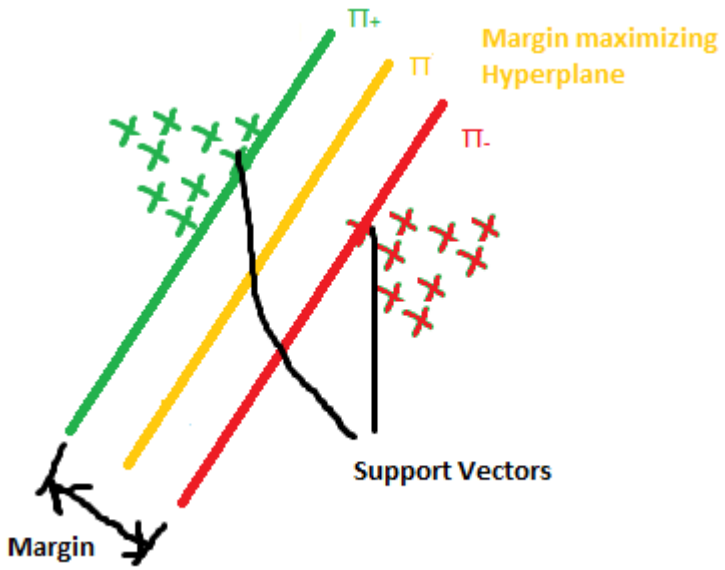
For classification we will use π, the margin maximizing hyper plane.

Alternative Geometric Intuition of SVM: Using Convex hulls:



Build smallest convex hulls for each set of data points, find the shortest line connecting these hulls, and draw a perpendicular bisector to this line to get the margin maximization hyper plane.

29.2 Mathematical derivation



Find hyper plane that does margin maximization; Let this hyper plane be $W^TX + b$, where W is perpendicular to the hyper plane.

For π^+: $W^TX + b = 1$

For π^-: $W^TX + b = -1$

Margin = $2/||W||$

Objective: Find $(W^*, b^*) = \text{argmax}\ (W,b)\ 2/||W||$

For $y_i\ (W^TX + b) >= 1$ (2 cases: +1 x +1 and -1 x -1) this is the constraint

This works very well for linearly separable data points.

If the data points are mixed to some extent or some data points lie in the margin or some positive data points lie on negative side of the hyper plane. We will never find a solution for these cases. The above constraint imposes a hard margin on the SVM model. Thus the margin needs to be relaxed.

Introduction of a new variable which tells how far the data point is in the opposite direction of the hyper plane. This variable introduced is ξ_i, which is equal to zero if the data point is on the correct direction of the hyper plane and is equal to the distance from the hyper plane if it is on the other side.

$(W^*, b^*) = \text{argmin}\ (W,b)\ ((||W||/2) + C * (1/n)\ \text{sum}(\xi_i))\ |\ y_i\ (W^T X + b) >= 1 - \xi_i$ for all i

This is an objective function which says to maximize the margin and minimize errors.

As C increases, tendency to make mistakes decreases, overfitting on training data, high variance. As C decreases underfitting occurs, high bias. The above objective formulation is for soft margin optimization.

29.3 Why we take values +1 and -1 for support vector planes

It does not matter, we can take +k and –k and also want both the support vector planes be equidistant from the margin maximizing hyper plane. +1 and -1 are chosen for mathematical convenience.

29.4 Loss function (Hinge Loss) based interpretation

Logistic Regression: Logistic loss + reg

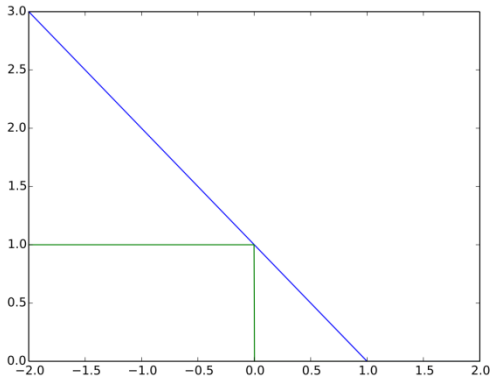
Linear Regression: Linear loss + reg

SVM: Hinge loss + reg

$Z_i = y_i\ f(x_i) = y_i\ (W^T X_i + b)$

For 0-1 loss: if $Z_i > 0$ is correctly classified; if $Z_i < 0$ is incorrectly classified

Hinge Loss:



If $Z_i >= 1$, hinge loss = 0;

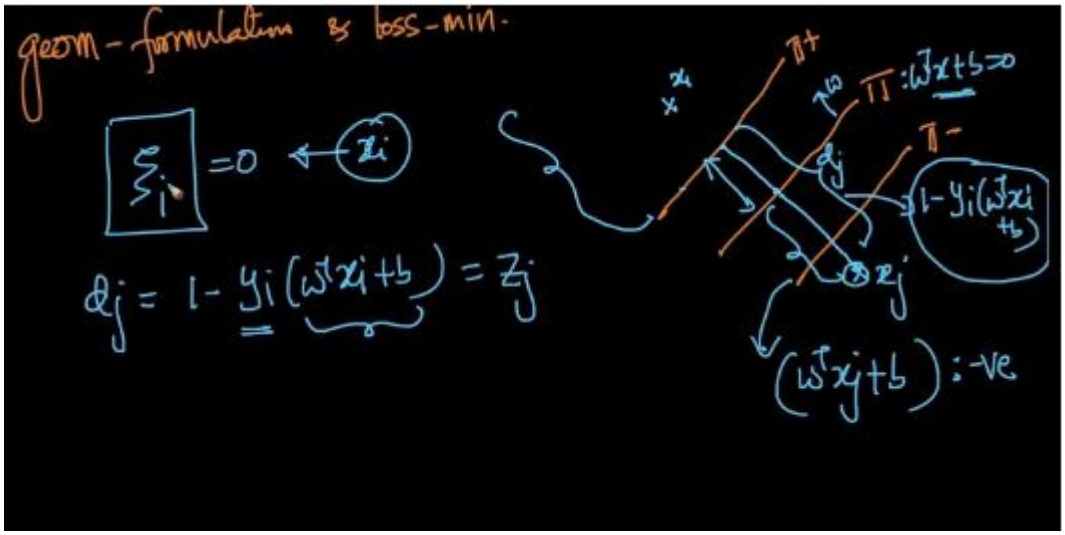
If $Z_i <= 1$, hinge loss = $1 - Z_i$;

Hinge loss = $\max(0, 1 - Z_i)$

Geometric formulation:

$\xi_i = 0$ for correctly classified data points

For incorrectly classified data points:



ξ_i = dist for x_j to the correct hyper plane = max(0,1- Z_j)

Soft-SVM: min(W,b) $||W||/2 + C \text{ sum}(\xi_i)$ such that $(1- y_i (W^T X_i + b)) <= \xi_i$

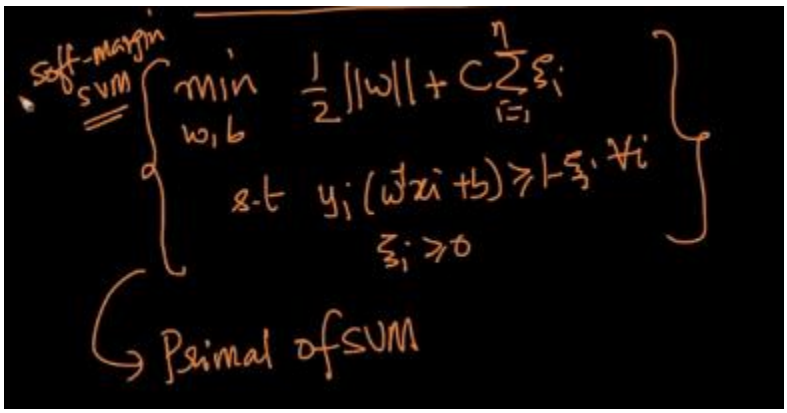
C increases - Overfitting

Loss-minimization: min (W,b) $\text{sum}(\max(0, 1-y_i(W^T X_i + b)) + \lambda||W||^2$

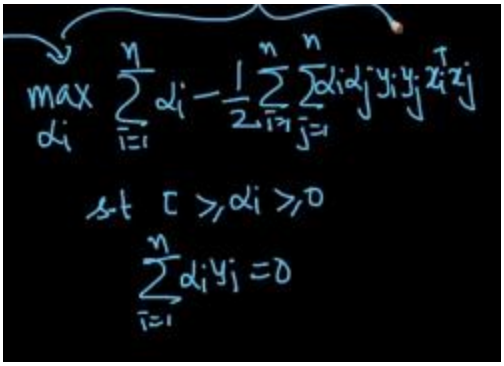
λ increases – Underfitting

(C when multiplied with loss function and λ for regularizing term)

29.5 Dual form of SVM formulation

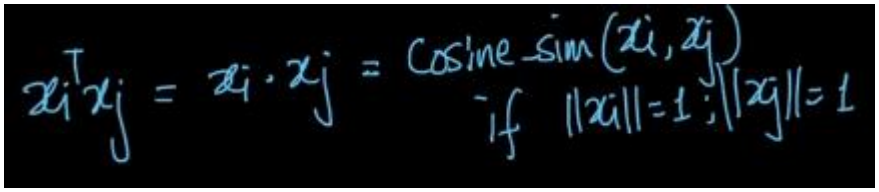


Dual form:



$\alpha_i > 0$ for support vectors and $\alpha_i = 0$ for non support vectors

The dual formulation of SVM does not depend only on Support Vectors.



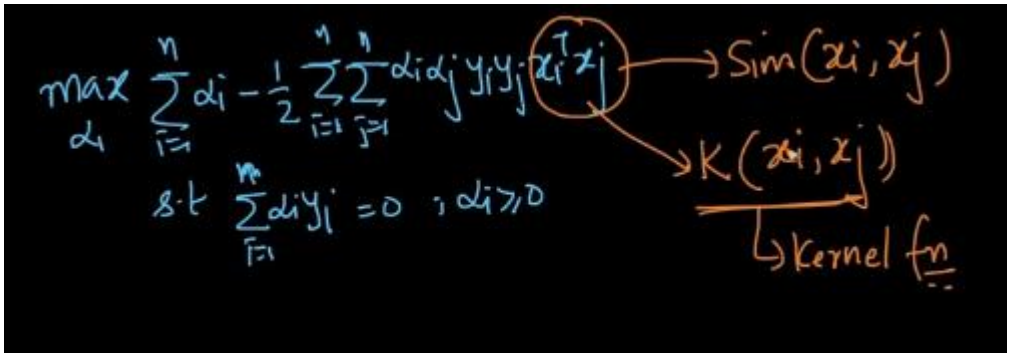
SVM can include similarity between data points using the Dual form.

K - Kernel function tells about similarity about the two data points.

Link: http://cs229.stanford.edu/notes/cs229-notes3.pdf

29.6 Kernel Trick

We have:



Without kernel trick the formulation is called as Linear SVM; Task: find margin maximizing hyper plane; Results look similar for Linear SVM and Logistic Regression;

A simple hyper plane cannot separate all types of data clusters;

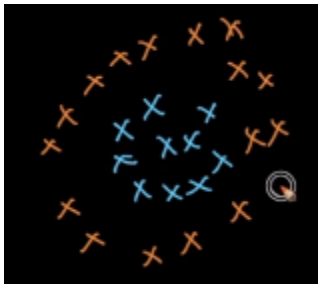
Logistic regression works on transformed space of data points;

Kernel-SVM can solve non-linearly separable datasets;

Kernel logistic regression also exists;

29.7 Polynomial Kernel

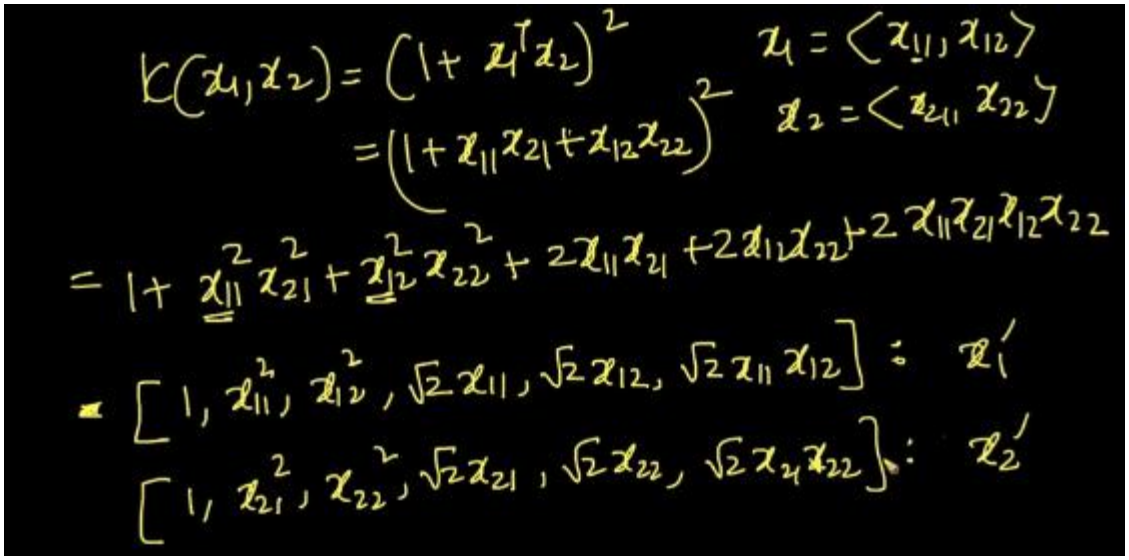
For dataset such as:



The dataset is transformed to square of features during Logistic regression modeling;

Polynomial kernel: $K(x_1, x_2) = (x_1^T x_2 + c)^d$;

Quadratic kernel:



Kernelization can be thought of application of feature transformation internally;

In logistic regression feature transformation is performed explicitly;

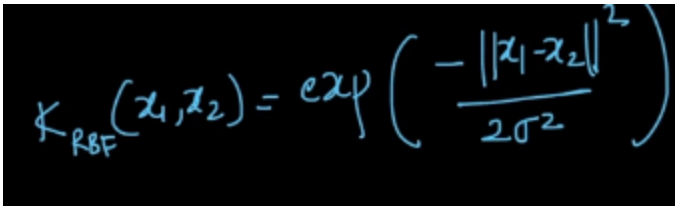
Mercer's theorem: Kernel trick converts d dim dataset to d' dim data set s.t. d' > d;

This will cater for non-linearity in the dataset;

Finding the kernel is the challenge;

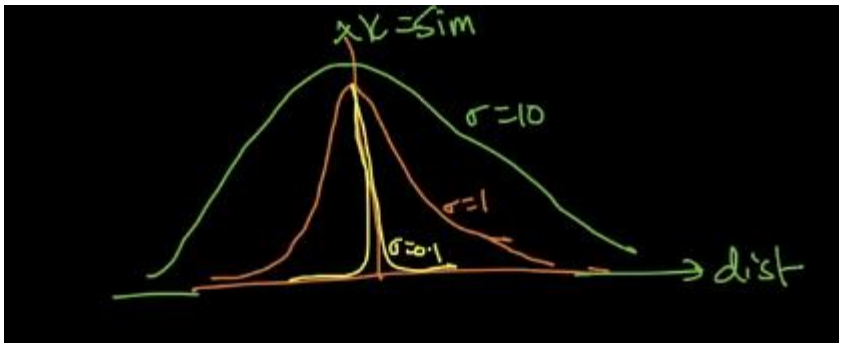
29.8 RBF-Kernel

General purpose and most popular Kernel: Radial Basis Function:



Numerator: distance squared; sigma is a hyper parameter for RBF kernel

1. As distance increases, $K(x_1, x_2)$ decreases and reaches 0
2. As sigma reduces, the tolerance to distances reduces, i.e. data points need to be very near to have similarity values non-zero



Dissimilarity : Distance :: Similarity : Kernel

As sigma increase, variance increases. Sigma increment is similar to increment in K.

Qs: How RBF Kernel is related to KNN? How is SVM related to Logistic Regression? How is Polynomial Kernel related to feature transformations?

RBF-SVM is similar to KNN because of sigma;

KNN: Stores all k-pts;

RBF-SVM: only require Support Vectors;

RBF-SVM is an approximation to Kernel;

We have C and Sigma as hyper parameters;

29.9 Domain specific Kernels

We have seen Polynomial and RBF kernels: RBF-SVM ~ KNN and RBF kernel is a general purpose kernel; Kernel-trick ~ Feature Transformation – Domain specific

String Kernels: Text classification

Genome Kernels: Genome prediction

Graph-based Kernels: Graph problems

RBF can be used as a general purpose kernel;

29.10 Train and run time complexities

Train by using SGF,

Use Sequential Minimal Optimization (SMO) for specialized algorithm (dual formulation)

Ex: libSVM

Training Time: $O(n^2)$ for kernel SVMs

SVMs are typically not use when 'n' is large

Run Time: O(kd); k – Number of Support Vectors, d - dimensionality

29.11 nu-SVM: control errors and support vectors

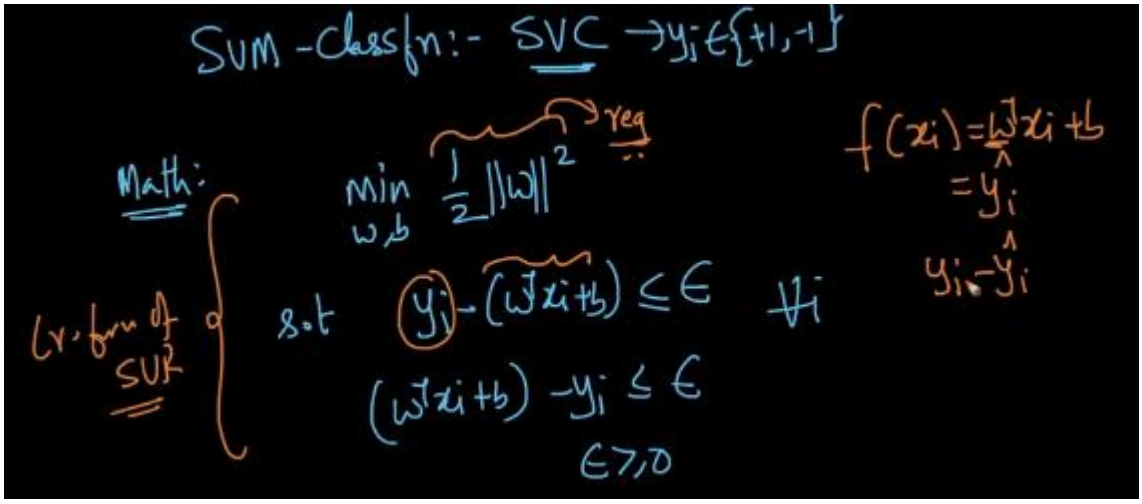
C-SVM: C>=0

nu-SVM: nu belongs to [0,1]

Fraction of errors <= nu; fraction of Support Vectors >= nu

Nu is upper bound on fraction of errors and lower bound on number of Support Vectors

29.12 SVM Regression



Epsilon is the hyper parameter;

With kernels we can fit non-linear datasets;

If epsilon is less; errors are low, overfitting increases;

RBF-Kernel SVR: similar to KNN Regression where nearest data points are calculated and a mean of the values of data points is calculated;

Links: https://alex.smola.org/papers/2004/SmoSch04.pdf, https://youtu.be/kgu53eRFERc

29.13 Cases

Feature Engineering: Kernelization

Decision Surfaces: Non- Linear surfaces

Similarity or distance function: Easily kernelizable

Interpretability and feature importance: no provisions with kernel SVM

Outliers have little impact as Support Vectors are important;

- RBF with a small sigma can be impacted with outliers as in case of kNN with small k

Bias Variance: As C increases – overfitting, high variance;

As C decrease – underfitting, high bias

Large dimensionality: SVMs perform well with large d

If dataset size or number of Support Vectors is large, then Support Vectors are not zpreferred; Instead Logistic Regression is used

29.14 Code Sample

https://scikit-learn.org/stable/modules/svm.html

29.15 Revision Questions

1. Explain About SVM?
2. What is Hinge Loss?
3. Dual form of SVM formulation?
4. What is Kernel trick?
5. What is Polynomial kernel?
6. What is RBF-Kernel?
7. Explain about Domain specific Kernels?
8. Find Train and run time complexities for SVM?
9. Explain about SVM Regression. ?

30.1 Questions and Answers

1. Give some situations where you will use an SVM over a RandomForest Machine Learning algorithm and vice-versa. (https://datascience.stackexchange.com/questions/6838/when-to-use-random-forest-over-svm-and-vice-versa)
2. What is convex hull?(https://en.wikipedia.org/wiki/Convex_hull)
3. What is a large margin classifier?
4. Why SVM is an example of a large margin classifier?
5. SVM being a large margin classifier, is it influenced by outliers? (Yes, if C is large, otherwise not)
6. What is the role of C in SVM?
7. In SVM, what is the angle between the decision boundary and theta?
8. What is the mathematical intuition of a large margin classifier?
9. What is a kernel in SVM? Why do we use kernels in SVM?
10. What is a similarity function in SVM? Why it is named so?
11. How are the landmarks initially chosen in an SVM? How many and where?
12. Can we apply the kernel trick to logistic regression? Why is it not used in practice then?
13. What is the difference between logistic regression and SVM without a kernel? (Only in implementation – one is much more efficient and has good optimization packages)
14. How does the SVM parameter C affect the bias/variance trade off? (Remember C = 1/lambda; lambda increases means variance decreases)
15. How does the SVM kernel parameter sigma^2 affect the bias/variance trade off?
16. Can any similarity function be used for SVM? (No, have to satisfy Mercer's theorem)
17. Logistic regression vs. SVMs: When to use which one? (Let n and m are the number of features and training samples respectively. If n is large relative to m use log. Reg. or SVM with linear kernel, if n is small and m is intermediate, SVM with Gaussian kernel, if n is small and m is massive, create or add more features then use log. Reg. or SVM without a kernel)
18. What is the difference between supervised and unsupervised machine learning?

External Resource: https://www.analyticsvidhya.com/blog/2017/10/svm-skilltest/

Chapter 31: Decision Trees

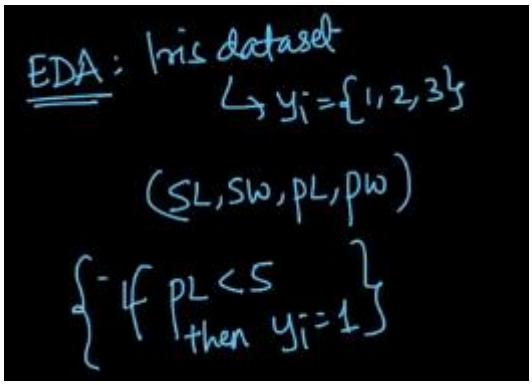
31.1 Geometric Intuition of decision tree: Axis parallel hyper planes

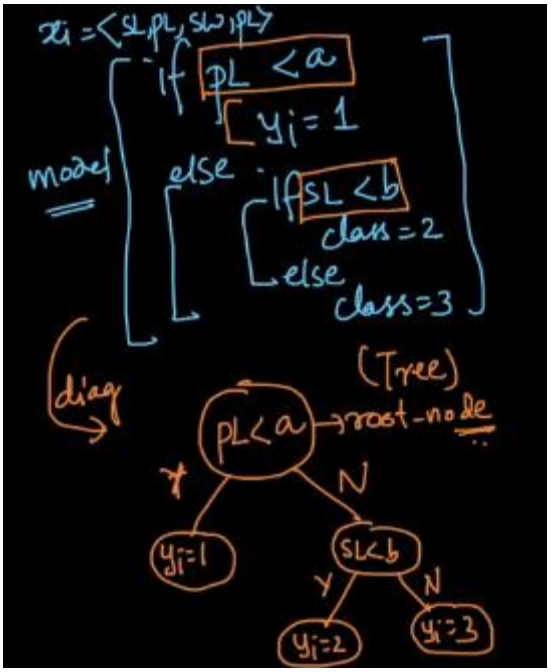
Classification and Regression methods:

KNN	Instance based method
Naïve Bayes	Probabilistic model
Logistic Regression	Geometric model: Hyper plane based separation
Linear Regression	Geometric model: Hyper plane based separation
SVM	Geometric model: Hyper plane based separation

Decision Trees: Nested If – else classifier

Ex:



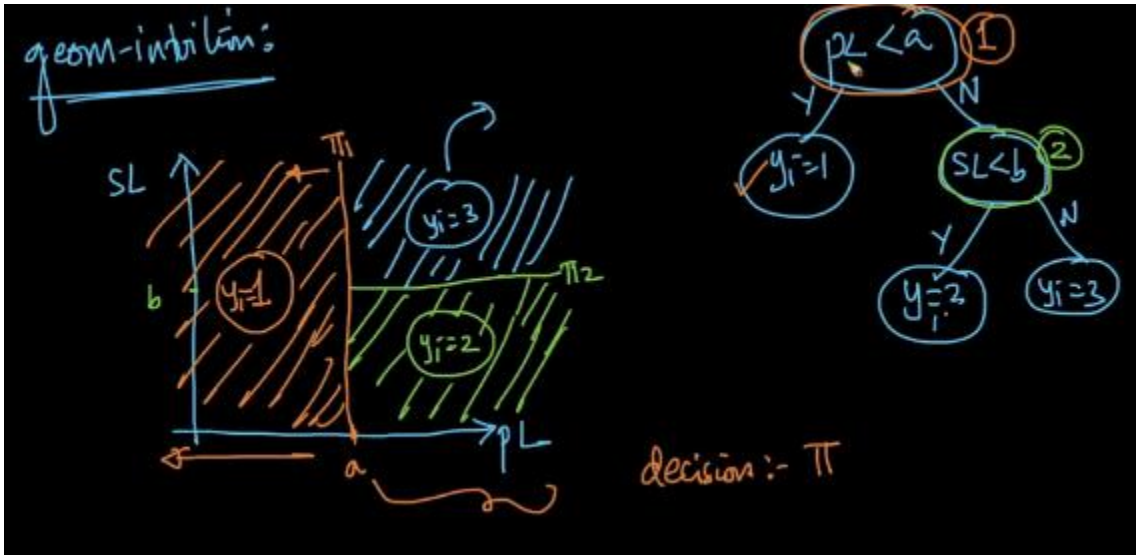


Root – node: First node

Leaf node: Terminating node

Non- Leaf node: Decision nodes

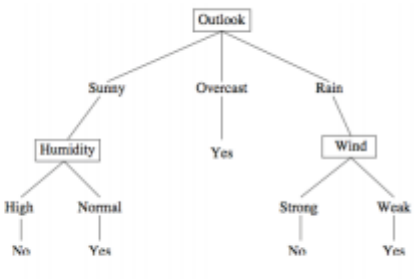
Decision Trees are extremely interpretable



All hyper planes are axis parallel;

31.2 Sample Decision Tree

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No



All observations in the data set are perfectly described by the tree.

Given any query point predicting its class label is straight forward.

31.3 Building a Decision Tree: Entropy

Task: Given a dataset build a Decision Tree

Entropy: occurs in physics, Information Theory, etc.

Given random variable y with k instances y1, y2, y3, …., yk

Entropy(y) = H(y) = - summ ($p(yi)*\log_2 (p(yi))$)

Let y = Play Tennis: y = {0, 1}

P(y=1) = 9/14, P(y=0) = 5/14

$H(y) = - (9/14) \log_2(9/14) - (5/14) \log_2(5/14)$

Case 1: 99% y+ and 1% y-: Entropy = -0.99lg 0.99 – 0.01 lg 0.01 = 0.0801

Case 2: 50% y+ and 50% y-: Entropy = - 0.5 log 0.5 – 0.5 log 0.5 = 1

Case 3: 0% y+ and 100% y-: Entropy = 0

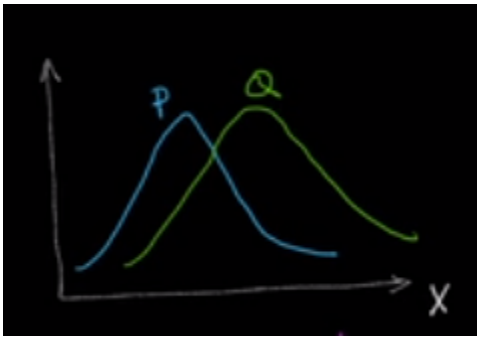
Entropy has maximum value when all classes are equally probable;

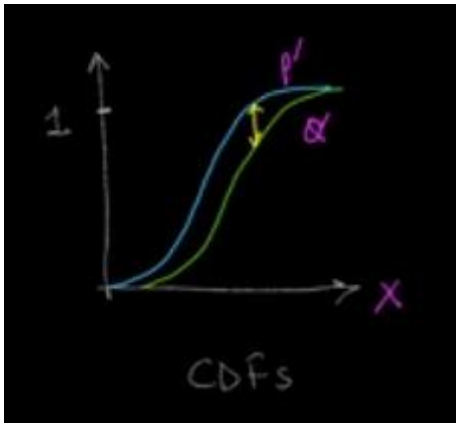
Entropy indicates a degree of randomness in the data.

31.4 KL Divergence

Kullback Leibler Divergence:

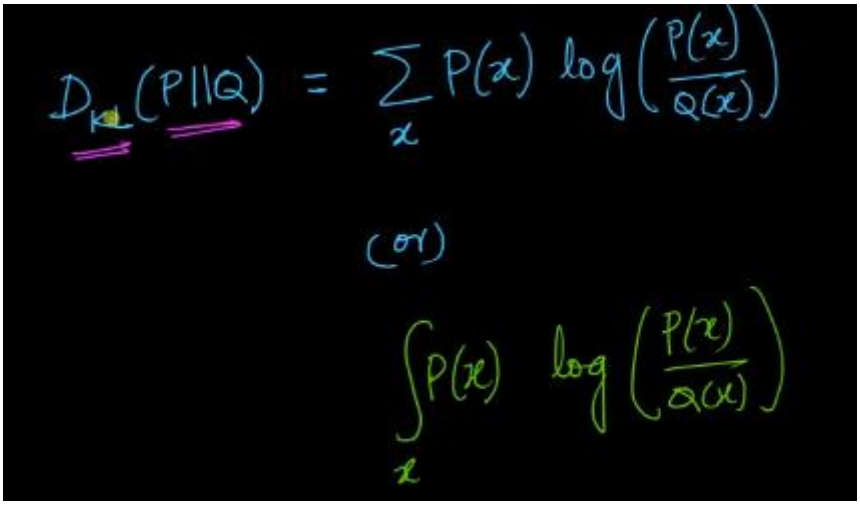
Two distributions P and Q:





Dist(P, Q): is small when two distributions are similar and closer;

KS statistic: Maximum gap between P' and Q': KS statistic is not differentiable thus cannot be used as a part of loss function;



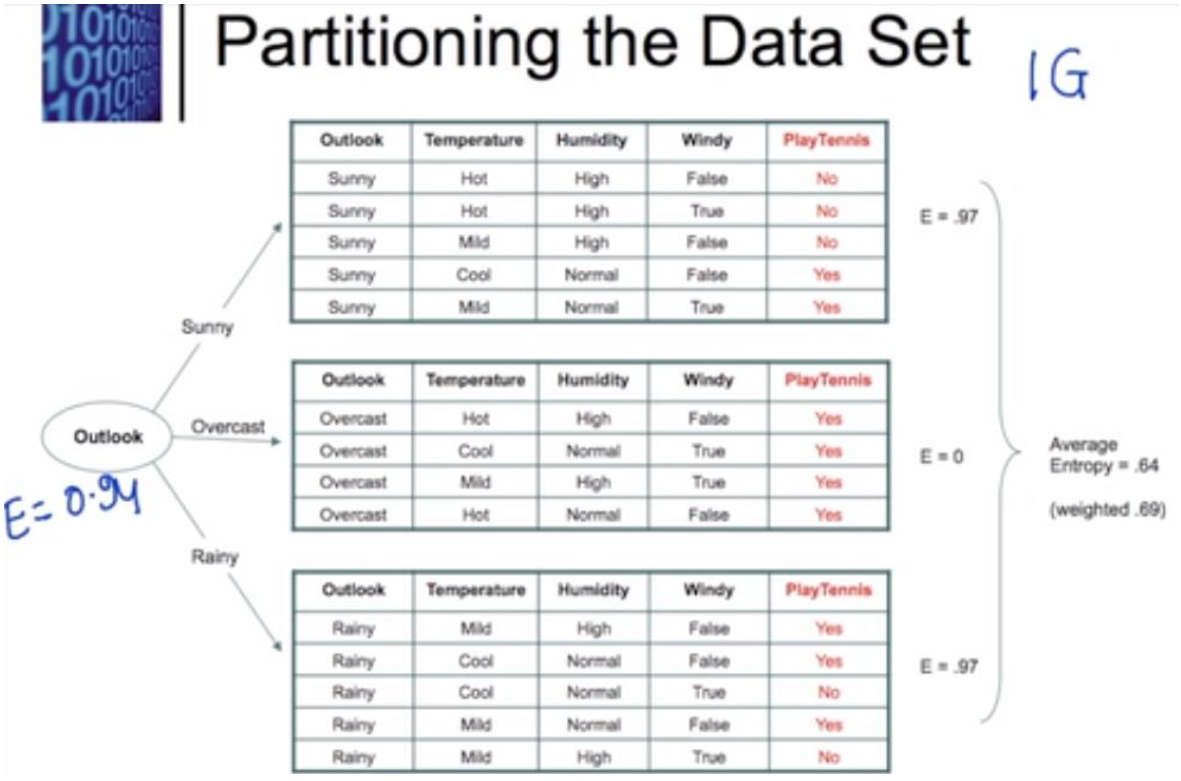
Summation or integration depends on type of random variable (discrete or continuous);

KL value will be increasing when there is dissimilarity in P and Q increasing;

KL statistic is differentiable

31.5 Building a Decision Tree: Information Gain

Information Gain = Entropy(parent) – Weighted Average of Entropy of child nodes



Outlook	Temperature	Humidity	Windy	PlayTennis
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Sunny	Mild	High	False	No
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	True	Yes

Outlook	Temperature	Humidity	Windy	PlayTennis
Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes

Outlook	Temperature	Humidity	Windy	PlayTennis
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Rainy	Mild	Normal	False	Yes
Rainy	Mild	High	True	No

E(Y, Outlook = Sunny) = - (2/5) *lg(0.4) – 0.6*lg(0.6) = 0.97

E(Y, Outlook = Overcast) = - 1*lg(1) - 0 = 0

E(Y, Outlook = Rainy) = - (3/5) *lg(0.6) – 0.4*lg(0.4) = 0.97

Information_Gain = Entropy(parent) – Weighted average of entropy of child nodes

= 0.94 – ((5/14)*0.97 + 4/14 * 0 + (5/14)*0.97)

31.6 Building a decision tree: Gini Impurity

Gini Impurity similar to entopy;

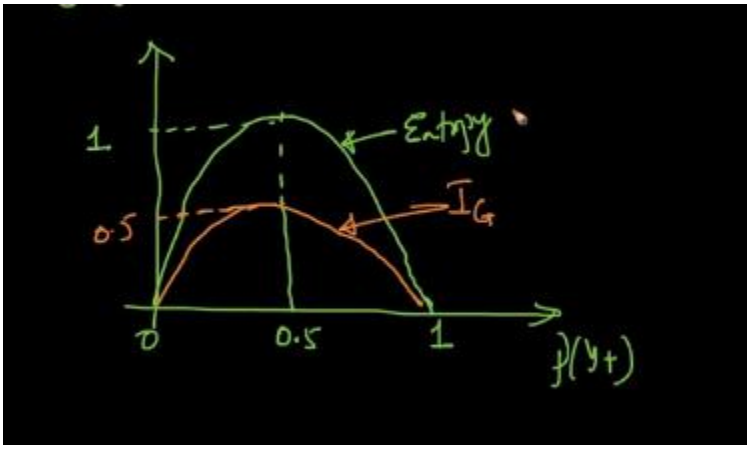
Advantages over Entropy:

$I_G(Y)$!= IG(Y)

$I_G(Y) = 1 – \text{sum } (p(y_i))^2$

Case 1: P(y+) = 0.5 and P(y-) = 0.5, $I_G = 1 – (0.5^2 + 0.5^2) = 0.5$

Case 2: P(y+) = 1 and P(y-) = 0, $I_G = 1 – (1 – 0) = 0$



Calculating logarithmic values is computational expensive that computing squares, thus as Entropy and Gini impurity behave similar to each other, Gini impurity is preferred for its compute efficiency.

31.7 Building a decision tree: Constructing a Decision Tree

Dataset: The play tennis case:

1. At top we have dataset with 9+ and 5-: H(Y) = 0.94
2. Outlook: H(Y, Sunny), H(Y, Overcast), H(Y, Rainy), weighted = 0.69, IG = 0.94 – 0.69 = 0.25
3. Temp: High, Mild, Cool: weighted = __, IG = __
4. Humidity: IG = __
5. Windy: IG = __

Result:

Stopping condition for growth of decision tree: Pure nodes or depth, if pure node not reached then use majority vote or mean;

Depth of tree is a hyper parameter determined by cross validation.

31.8 Building a decision tree: Splitting numerical features

Task: Split node: using Entropy or Gini Impurity and Information Gain

Categorical feature is straight forward;

For Numerical features, sorting of data points is done based on feature values: and numerical comparisons are done to split nodes;

Evaluate all possible threshold splits; Compute Information Gains;

31.9 Feature Standardization

Mean centering and scaling;

Threshold depends on order and does not depend on actual values in Decision Trees, No need for feature standardization

31.10 Building a decision tree: Categorical Features with many possible values

Ex: Pin code/ Zip code: Numerical but not comparable, thus these are taken as categorical features and there could be a lot of categories in each feature; which will result in data sparsity or tree sparsity

Transforming this into Numerical features will be useful:

> Calculate conditional probabilities of class labels given a certain categorical value

31.11 Overfitting and Underfitting

As depth increases, possibility of having very few data points at a node; possibility of overfitting increases and interpretability of the model decreases; as depth is less underfitting happens.

With Depth = 1 the decision tree is called a decision stump,

Depth is determined using Cross Validation

31.12 Train and Run time complexity

Train time: ~ O (n * log n * d)

Space at run time: ~ O (nodes)

Time at run time: ~ O(depth) = ~ O(1)

Decision Trees are suitable: Large data, small dimensionality, low latency requirement

31.13 Regression using Decision Trees

Information Gain for classification; MSE or MAE for regression;

y_i_hat = mean/ median at each node;

Take weighted errors for splitting, whichever feature gives lowest error is selected;

31.14 Cases

Imbalanced data impacts Information Gain calculations;

Large dimensionality increases time complexities

Categorical features avoid one hot encoding; Use response encoding or Label encoding or probabilistic encoding

Decision Trees cannot work with similarity matrix;

Multi-Class Classification: One versus Rest is not required as Entropy takes all categories while calculating them

Decision Surfaces: Non- Linear Axis parallel Hyper Cuboids

Feature Interactions: Between depths, F1 < 2 AND F2 > 5 are available or logical feature interactions are inbuilt to Decision Trees: Advantageous

Outliers: As depth is large, overfitting occurs, outliers make Trees unstable;

Interpretability: Nested if-else conditions; DTs are extremely interpretable;

Feature Importance: For every feature we get Information Gain or can sum up reductions in Entropy due to this feature. The more reduction, the more important;

31.15 Code Samples

Link: http://scikit-learn.org/stable/modules/tree.html

Visualize: Dataset using Graphviz

DTClassifier: Gini Impurity or Entropy

DTRegressor: MSE or MAE (mean/ median IG)

31.16 Revision Questions

1. How to Build a decision Tree?
2. What is Entropy?
3. What is information Gain?
4. What is Gini Impurity?
5. How to Constructing a DT?
6. Importance of Splitting numerical features?
7. How to handle Overfitting and Underfitting in DT?
8. What are Train and Run time complexity for DT?
9. How to implement Regression using Decision Trees?

Additional Material:

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model;

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.

- Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Chapter 32: Interview Questions on decision trees

32.1 Question & Answers

1. You are working on a time series data set. You manager has asked you to build a high accuracy model. You start with the decision tree algorithm, since you know it works fairly well on all kinds of data. Later, you tried a time series regression model and got higher accuracy than decision tree model. Can this happen? Why?(Refer :https://www.analyticsvidhya.com/blog/2016/09/40-interview-questions-asked-at-startups-in-machine-learning-data-science/)
2. Running a binary classification tree algorithm is the easy part. Do you know how does a tree splitting takes place i.e. how does the tree decide which variable to split at the root node and succeeding nodes?(Refer:https://www.analyticsvidhya.com/blog/2016/09/40-interview-questions-asked-at-startups-in-machine-learning-data-science/)

https://vitalflux.com/decision-tree-algorithm-concepts-interview-questions-set-1/

Chapter 33: Ensemble Models

33.1 What are ensembles?

In Machine Learning, Multiple models are brought together to build a powerful model.

The multiple models may individually perform poorly but when combined they become more powerful, this combination to improve performance of several baseline models is called ensemble.

Ensemble (four of the Ensemble methods are as following):

- Bagging (**B**ootstrapped **Ag**gregation)
- Boosting
- Stacking
- Cascading

High performing, Very powerful, Very useful in real world problems;

Key aspect: the more different the base line models are, the better they can be combined

Problem: Models can be thought of as experts in different aspects of the problem and when brought together we will get a better solution

33.2 Bootstrapped Aggregation (Bagging) Intuition

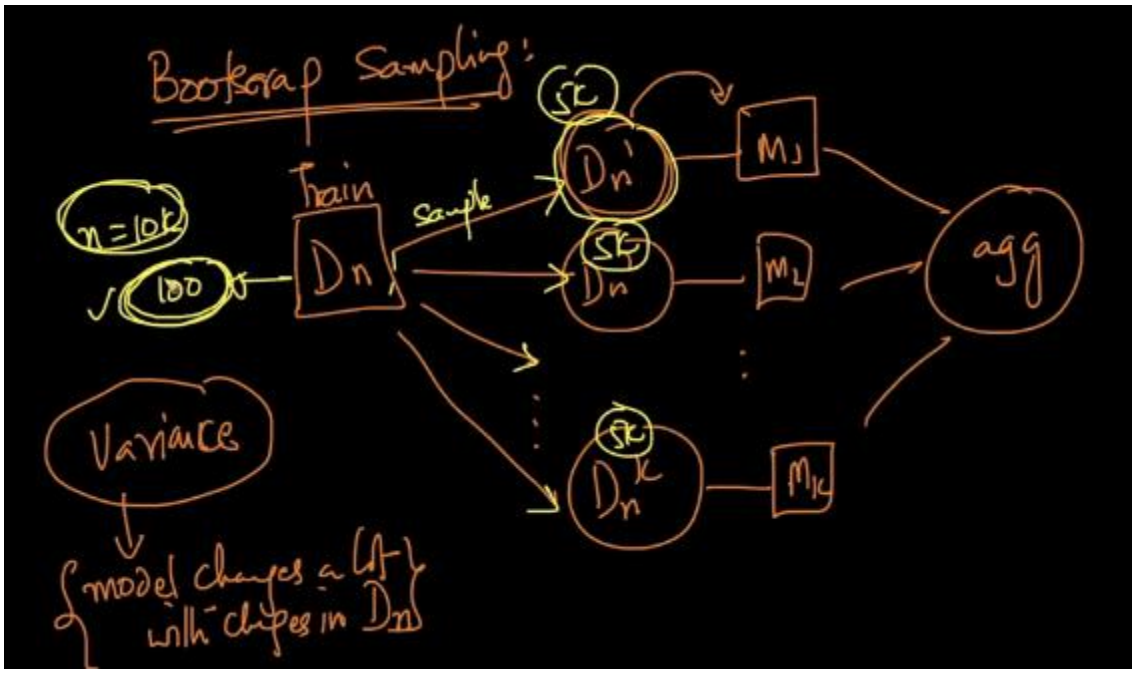
Sample with replacements and train models on different samples: Bootstrapping

Aggregation combining these different training models with a majority vote;

At run time, the query point is passed to all the models and a majority vote is applied

Classification: Majority, Regression: Mean/Median

None of the models are trained on the whole dataset;



Bagging can reduce variance in modeling due to aggregation without impacting bias;

Bagging: Take Base models having low bias and high variance and aggregate them to get a low bias and reduced variance model

Decision Trees with good depth are low bias and high variance models: Random Forest

33.3 Random Forest and their construction

Bagging with Decision Trees + Sampling (Feature Bagging)

Bootstrap sampling = Row sampling with replacement: Create k samples;

Column Sampling = select a subset of features randomly

$X_{n \times d}$ to $X_{m \times d'}$ where m<n and d' < d; m and d' are also different between samples;

We will have k Decision Trees trained on these k samples and Aggregation (majority weight or mean/median) is applied on the output of the k base learners (DTs);

The data points that are removed during selection for each sample (out of bag points) can be used for cross validation for the corresponding model (out of bag error is evaluated to understand performance of each base learner)

Random Forest: Decision Tree (reasonable depth) base learner + row sampling with replacement + column sampling + aggregation (Majority vote or Mean/Median)

Tip: Do not have a favorite algorithm

33.4 Bias – Variance tradeoff

Random Forest: low bias (due to base learners)

As number of base learners increases, variance decreases (due to aggregation)

Bias (individual models) ~ Bias (Aggregation)

d' / d = column sampling ratio and m/n = row sampling ratio

If these two ratios are decreased, variance decreases

These ratios are fixed initially and the number of base learners is determined through hyper parameter tuning (cross validation)

33.5 Train and run time complexity

K base learners;

Train time complexity: O(n * lg n * d * k); Trivially parallelizable, each base learner can be trained on different core of the computer;

Run time complexity: O(depth * k): low latency

Space complexity at run time: O(Decision Trees * k)

33.6 Bagging: code sample

Code: sklearn.ensemble.RandomForestClassifier()

33.7 Extremely randomized trees

For numerical features we apply thresholds (using sorting) to get Information Gain;

With extremely randomized trees, random selection of threshold values is done to check for Information Gain; In Decision Trees, we check for each of the values as threshold

Extreme Trees: Col sampling + Row sampling + Randomization when selecting thresholds + Decision Trees + Aggregation

Randomization is used to reduce variance; bias does not increase due to randomness;

33.8 Random Forest: Cases

Random Forest: Decision Tree + (Row samp. + Col samp + agg) (used to reduce variance)

1. Decision Trees cannot handle large dimensionality; with categorical features having many categories (fails even with one hot encoding), DTs and Random Forests have similar cases except for bias – variance tradeoff and feature importance
2. Bias – Variance Tradeoff of Random Forest is different from Decision Trees;

   In Decision Trees we change depth;

   In Random Forests: # of base learners is a hyper parameter and each decision tree has sufficient depth

3. Feature Importance:
   In DT, Overall reduction in Entropy of I_G because of this feature at various levels in the DT (single DT)
   In RF, overall reduction is checked across all base learners for feature importance

33.9 Boosting Intuition

Bagging: High variance and low bias models: with randomization and aggregation to reduce variance

Boosting: take low variance and high bias models: use additive combining to reduce bias;

Given a dataset:

Stage 0: Train a model using the whole dataset: This model should have high bias and low variance: DT with shallow depth; Large Train error;

Subtract: y_i – the output of this Decision Tree;

Stage 1: In stage 1, we train a model on error of the previous model

$F1(x) = a1\ h0(x) + a1\ h1(x)$

Stage k: $F_k(x) = \text{summ}(\text{alpha\_i} * h_i(x))$: additive weighted model

Each of the models at each stage is trained to fit on the residual error at the end of the previous stage

Methods: Gradient Boosted DT and Adaptive Boosting

33.10 Residuals, Loss functions and gradients

$F_k(x)$ = summ (alpha_i * $h_i(x)$)

Alpha_i coefficients and h_i residuals;

Residual: $err_i = y_i - F_k(x)$

$Model_{k+1}$ is trained on $\{x_i, err_i\}$

ML models are trained by minimizing loss functions:

Logistic loss – classification

Linear regression – Square loss

SVM – Hinge loss

$L(y_i, F_k(x))$: loss is defined on target and prediction at end of stage k

Let the problem be a regression problem: The loss function is say squared loss

$L(y_i, F_k(x)) = (y_i - F_k(x))^2$

Derivative of L wrt to F is $-1*2*(y_i - F_k(x)) = 2*$ residual of last stage

- Negative gradient of loss function at the end of stage k is proportional to residual at that stage; negative gradient ~ pseudo residual (this allows to use any loss functions)
- RandomForest are limited with loss functions, Gradient Boosted DTs permit use of any loss function

Link: https://youtu.be/qEZvOS2caCg

For non squared loss functions: Is pseudo residual equal to negative gradient?

Log loss: negative gradient = probability estimate – actual class

But pseudo residuals cannot be interpreted as residual / error always

33.11 Gradient Boosting

Input: training set $\{(x_i, y_i)\}_{i=1}^n$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
    1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^n$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

33.12 Regularization by shrinkage

As number of base models increase, overfitting occurs with increase in variance;

$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x), \quad 0 < \nu \leq 1$$

This controls overfitting; shrinkage is a hyperparameter;

33.13 Train and Run Time complexity

Decision Trees: Train: O(n * lg n * d*M)

Random Forests are trivially parallelizable;

While GBDTs are trained sequentially;

Run time complexity: O(depth * M)

Space complexity at run time: O(store each tree + gamma)

GBDTs used for low – latency applications

They can minimize any loss function;

33.14 XGBoost: Boosting + Randomization

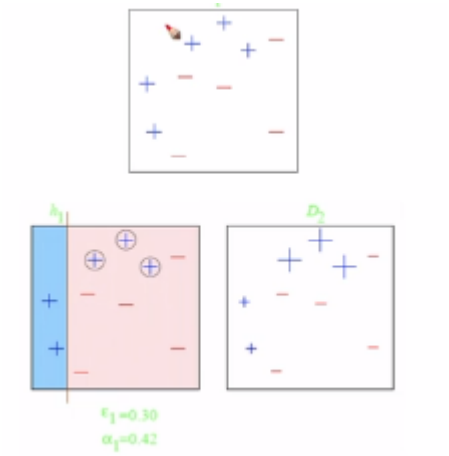
Extension of GBDT and RF

33.15 AdaBoost: geometric intuition

Used in Computer Vision for face detection;

In GBDT we used Pseudo Residuals;

Adaptive boosting: At every stage you are adapting to errors that were made before; more weight is given to misclassified points



Adaboost is highly prone to outliers;

33.16 Stacking models

An ensemble learning meta-classifier for stacking;

Individual classification models are trained based on the complete training set; then a meta-classifier is trained on the outputs – Meta features of the individual classification models in the ensemble. The individual models should be as different as possible.

Train all models independent of each other; given a data point query it is passed through all models and all the outputs are passed through a meta-classifier;

Using outputs of base models as Meta features for a new classifier;

Real World: Stacking is least used on real world problems due to its poor latency performance

33.17 Cascading classifiers

Used when cost of making a mistake is high; Sequential model definition each model working to classify datapointso ne after the other; whichever data points are perfectly classified with high probability by a model are not shown to the next model; generally at the end of all the models a human is placed to make predictions on the data points the cascaded models are unsure about;

Used for fraud detections;

33.18 Kaggle competitions vs Real World Case Studies

Bagging, Boosting, Stacking, Cascading;

Kaggle competitions care about only one metric;

Due to this complex ensembles are generally experimented with and these models are not useful for low latency, training time and interpretability;

- Brilliant ideas for feature engineering are available;

33.19 Revision Questions

1. What are ensembles?
2. What is Bootstrapped Aggregation (Bagging)?
3. Explain about Random Forest and their construction?
4. Explain about Boosting?
5. What are Residuals, Loss functions and gradients?
6. Explain about Gradient Boosting?
7. What is Regularization by Shrinkage?
8. Explain about XGBoost?
9. Explain about AdaBoost?
10. How do you implement Stacking models?
11. Explain about cascading classifiers?

Module 5: Feature Engineering, Productionization and deployment of ML models

Chapter 34: Featurization and Feature Engineering

34.1 Introduction

Most important process in ML is feature engineering;

> Data can be in the form of Text, categorical, numerical, time series, image, database tables, Graph-data, etc.

Text data: BOW, TFIDF, AVGW2V, TFIDFW2V

Categorical data: One-hot Encoding, response encoding

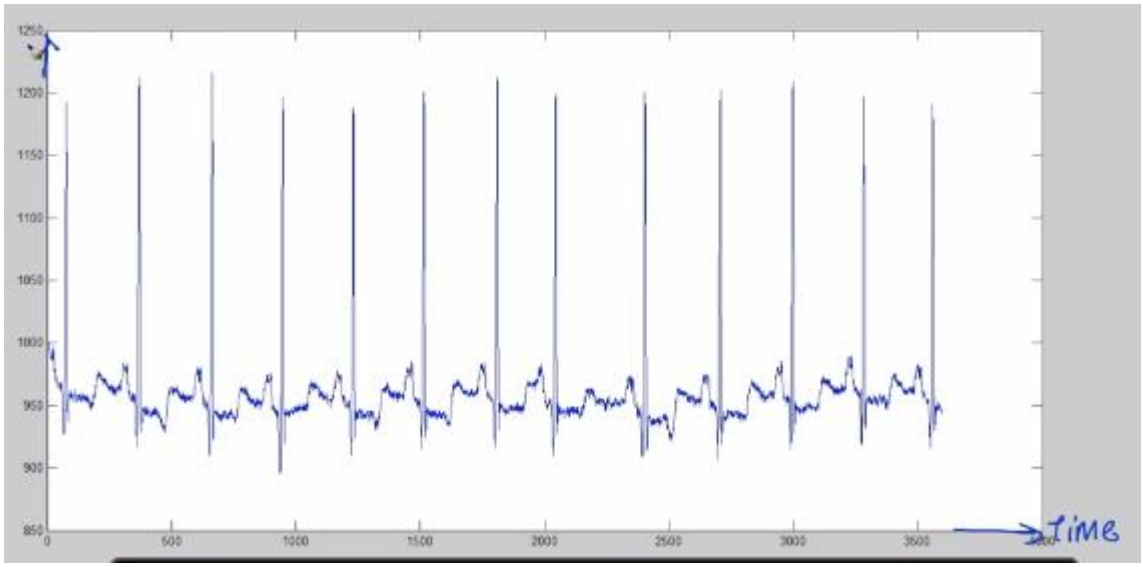
Time series (ex: heart rate): transformed to frequency domain, etc.

Image data (ex: face detection): transformed to arrays

34.2 Moving Window for Time Series Data

Simplest featurization method for time series data;

Example: ECG



Moving window takes a time split of a width of time;

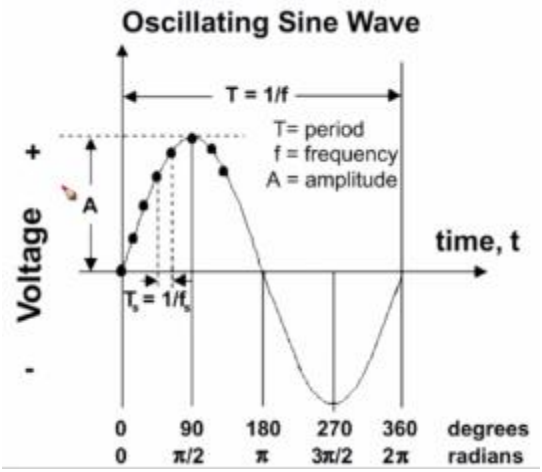
Featurization techniques are decided based on damage; some standard features extracted from window of the time series are Mean, Median, Max and min, Max - min, Standard deviation, quantile, number of local minima or maxima, mean crossings or zero crossings all of this for each window;

The features are problem specific;

1. Window width – hyper parameter
2. Features useful: Generate dataset; data points correspond to different windows

34.3 Fourier decomposition

A Method to represent time series data;



T = Time taken for each oscillation

Freq = 1/T (Hz)

Example: Heart rate: 60 – 100 BPM = 1 – 1.6 BPS = 1 – 1.6 Hz

Phase: Difference in angle at which different waves start;

We can have daily shopping patterns or weekly shopping patterns or annual patterns

Given a composite wave: It can be decomposed into multiple sine waves, different sine waves have different time period, and different peaks: this is transformed onto frequency domain; This process is called as Fourier transformation;

Useful when there is a repeating pattern;

Vectorization using important frequency and amplitudes;

Accelerometer: electro-mechanical device used to measure acceleration forces. They use Piezo electric effect: where when the crystal is acted upon by an accelerative force an EMF is generated in the circuit.

34.4 Deep Learning Features: LSTM

For each problem: a specific set of features are designed prior to Deep Learning; A set of features designed for a problem did not work for another problem

Deep Learnt Features: Used for time series using LSTM

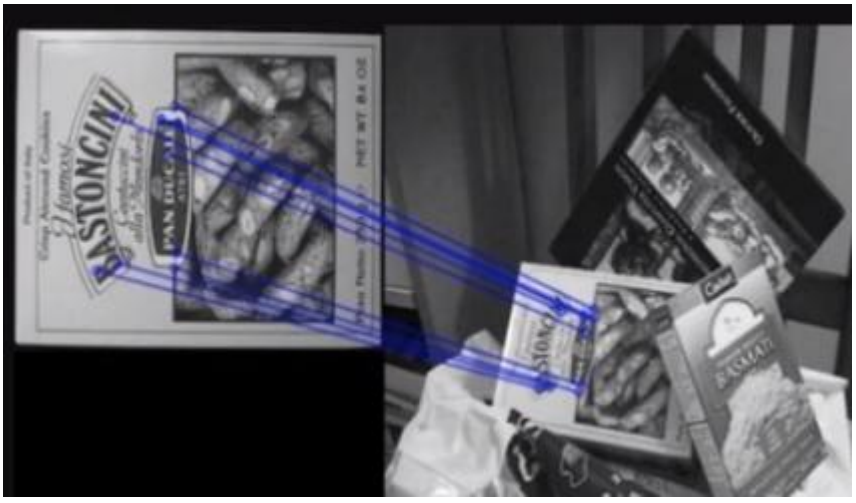
34.5 Image histogram

Color histogram: Each pixel has 3 color values; for each color plot a histogram (0 to 255) and vectorize it; Different objects have distinct color attributes, sky and water come in blue, human faces have cream, wood brown, metal grey, etc.

Edge histogram: At the interface of color separation; region based edge orientation;

34.6 Key points: SIFT

Scale Invariant Feature Transforms;



Detects key points or corners of object and creates a vector for each key point;

These key features are detected and extracted for predictions;

SIFT has scale invariance and small rotation invariance

34.7 Deep Learning: CNN

Time series data: LSTM

Images: CNN

34.8 Relational data

Data stored in multiple tables and connected through relations:

Ex: Shopping database

Feature examples: Count, Budget

34.9 Graph data

Example: Facebook: User Social Graph

Task: Recommend new friends

Features: Number of paths different nodes of the Graph, Number of mutual friends

34.10 Indicator variables

Converting a feature into indicator variables:

Height: Numerical to indicator variable, example: binary: 1 if > 150, 0 else;

Country: If country == India Or USA return 1 else 0

34.11 Feature Binning

Extension to indicator variable: Multi class encoding:

Height: If H<120 return 1 elif H<150 return 2 elif H<180 return 3 else return 4

Task: Predict gender given, height, weight, hair length, eye color

return 1 if male and return 0 if female

Table: Converting height numerical data into categorical using binning:

Train a simple decision tree using height:

If H < 130cm $y_i = 0$ (thresholds determined using Information Gain)

Thresholds found using DT

Decision Tree Based Feature Binning

34.12 Interaction Variables:

Example: 1. if h<150 and w<=60 kgs return 0: 2 way interaction

2. h * w or $w*h^2$: math operations 2 way interaction here

3. h < 150, w<65 and hL > 75 return 0: 3 way interaction

Given a task find interaction features:

Using Decision Trees: Create new features using all leaf nodes;

34.13 Mathematical transformations

X is a feature: log(X), sin(X), X^2, etc. can be used to transform the features;

Distribution of X can be inferred: Power law distributed can be logged;

34.14 Model specific featurizations

Say f_1 is power law distributed: when using Logistic Regression: transforming with log is beneficial;

3 features and a regression target: say we have f1 – f2 + 2 f3: Linear models are better; Decision Trees may not work here

If interaction of features are prevalent DT perform well;

With Bag of Words: Linear models tend to perform very well (Lr SVM and Logistic Regression);

34.15 Feature Orthogonality

The more different the features (Orthogonal relationship) are, the better the models will be;

If there are correlations (less orthogonality) among the features, the overall prediction performance of the model on the target will be poor;

Create new features with errors of a model; easily over-fits;

34.16 Domain specific features

Reading research areas or literature to understand the best feature engineering methods as one technique will not fit all

34.17 Feature slicing

Split data into feature values and build different models on the splits separately;

Split criteria: feature values are different and sufficient data points are available;

Example: Customers buying Desktops and Mobile

34.18 Kaggle Winner Solutions

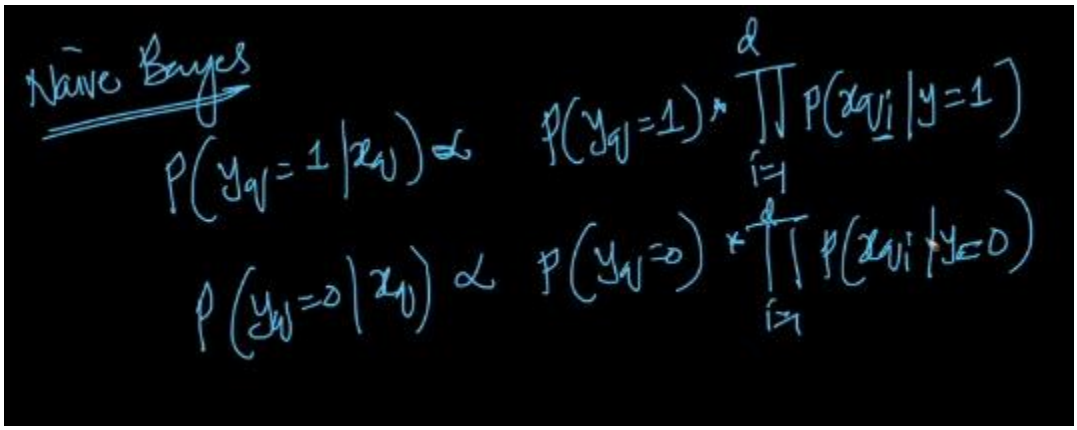
Feature engineering methods can be learnt from kaggle winner solutions,

And discussion of competitions

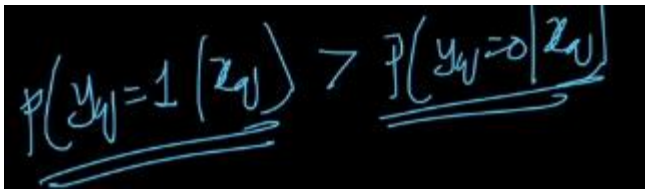
Chapter 35: Miscellaneous Topics

35.1 Calibration of Models: Need for calibration

Binary or 2 class classification problem: We might require understanding a probability score of belonging to one class;



If:



then class 1

These probabilities may not be exact probability; values between 0 and 1 can be given by sigmoid;

Calibration is required for computing the loss functions; these loss functions might depend on exact probabilities;

35.2 Calibration plots

Model f trained on a training dataset; For each data point this model will give an output:

Build a data frame of x_i, y_true and y_pred;

Sort this data frame in increasing order of y_pred;

Break table into chunks of size m;

In each chunk compute average of y_pred and y_true in the chunk; avg_y_pred and avg_y_true

Task of calibration: make avg_y_pred proportional to avg_y_true; an ideal model will result in a 45^0 line between avg_y_pred and avg_y_true;

35.3 Platt's Calibration / Scaling / Sigmoid Calibration

$P(y_q = 1 / x_q) = 1/ (1 + \exp(A\ f(x_q) + B)$

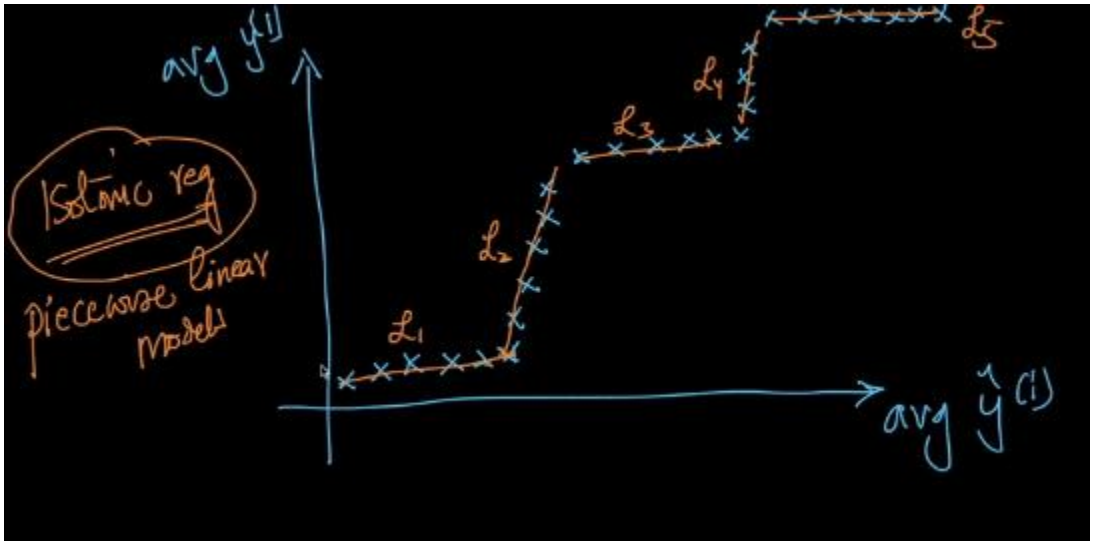
Platt scaling works only if the Calibration dataset distribution is similar to sigmoid function;

Isotonic calibration is generally applied: use case log loss

35.4 Isotonic Regression

Works even if calibration plot does not look like sigmoid function; if we fit a sigmoid function there will be a large error

Break the model into multiple linear regression models; piece wise linear models



Solve an optimization where minimize the gap between the gaps of the model and the actual value;

Large data → large Cross Validation dataset → large Calibration data → Isotonic Reg

Small data → Platt Scaling

35.5 Code Samples

Links: http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html

http://scikit-learn.org/stable/auto_examples/calibration/plot_calibration.html#sphx-glr-auto-examples-calibration-plot-calibration-py

35.6 Modeling in the presence of outliers: RANSAC

Random Sampling Consensus:

To build a robust model in the presence of outliers:

Sampling will reduce the presence of outliers in the training set

Apply the model and compute loss;

Determine outliers based on loss function and remove some outliers; repeat the process until satisfied; Reduction of dataset at each stage with removal of outliers based on random sampling, model training and loss computation;

35.7 Productionizing models

Link: http://scikit-learn.org/stable/modules/model_persistence.html

1. Using SKLearn: model persistence: storing in pickle format

   Using joblib.dump() from sklearn.externals.joblib

   Load using joblib.load()

2. Custom implementation:

   Store all of the parameters of the model to a file:

   For low latency applications; Use C/ C++ code for faster predictions;

   Store weights using data structures, Using CPU cache; RF and DTs with if else statements

Implementation is problem specific

35.8 Retraining models periodically

Time series data: Stock price prediction:

Model need to be retrained periodically to gather current trends and also to get around with new companies, etc.

Dataset changes and dropping model performance or retrain regularly;

35.9 A/B Testing

= Bucket testing or split run or controlled experiments;

Example: With respect to curing cold: If a new drug is found and need to be experimented for its effectiveness: A part of group of patients is administered with the new drug and another part of the group is administered with the old drug. Person with old drug/data are called control group and the group with new drug is called treatment group. The control and treatment groups are called A and B groups.

35.10 Data Science Life Cycle

1. Understand business requirements: define the problem and the customer

2. Data acquisition: ETL (extract transform and load), SQL

3. Data preparation: Cleaning and pre-processing

4. EDA: plots, visualization, hypothesis testing, data slicing

5. Modeling, evaluation and interpretation

6. Communicate resulys: clear and simple, 1-pager and 6 pagers

7. Deployment

8. Real-world testing: A/B testing

9. Customer/ business buy-in

10. Operations: retrain models, handle failures, process

11. Optimization: improve models, more data, more features, optimize code

35.11 Productionization and deployment of Machine Learning Models

35.7 Same topic

35.12 Live Session: Productionization and deployment of Machine Learning Models

--- Live sessions notes

35.13 Hands on Live Session: Deploy and ML model using APIs on AWS

--- Live sessions notes

35.14 VC dimension

Measure the potential of a class of models:

Linear Models, RBF SVM, Boosting algorithms, Neural Networks;

1. Practical approach: deploying and measuring the performance
2. Statistical approach: using Vapnik – Chevvonenkis (also build SVMs)

VC dimension is mostly used for research work not generally found in applied work;

Linear model: Decision Surfaces: Logistic Regression, Linear SVM;

VC dim (linear model) = maximum number of points that can be shattered by a linear model for all possible configurations = 3 (that are not collinear)

Link for further reading: https://en.wikipedia.org/wiki/VC_dimension

Theoretically, RBF SVM is powerful than all models as its VC dimension is infinity;

Practically, this did not apply due to limitations of assumptions;

Module 7: Data Mining (Unsupervised Learning) and Recommender Systems + Real – World Case Studies

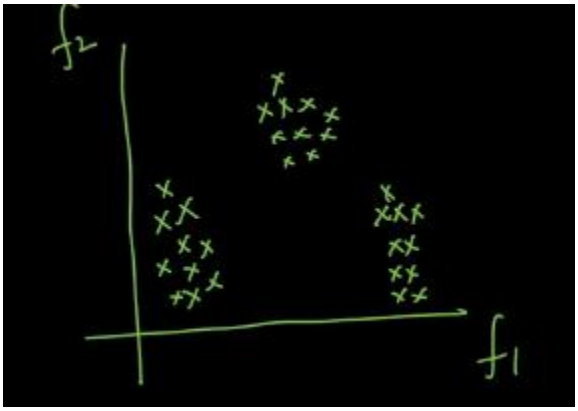
Chapter 43: Unsupervised Learning/ Clustering

43.1 What is Clustering?

Classification & Regression: Given a data point we have a target value;

Clustering: No target values; Task is to group similar data points;

Example:



Task: group similar points: we can have three groups here

Similarity: Points are close to each other in the same cluster and very far in different clusters

Measure of performance for Clustering Algorithms = ___

For classification and regression we had metrics such as precision, MSE, etc.

Clustering: We have k-Means, Hierarchical Clustering, DBSCAN

43.2 Unsupervised Learning

Classification and Regression: Supervised Learning: We had target variable to train the models

Semi- Supervised Learning: Dataset which has a small size of supervised data points and unsupervised data points;

43.3 Applications

Data Mining;

Ecommerce (group similar customers based on their location, income levels, purchasing behavior, product history)

Image segmentation: Grouping similar pixels, apply ML algorithm to do object detection

Review analysis (text): Manual labeling is time consuming and expensive; Clustering can be applied: into 10k groups based on word similarities syntactic and semantic similarity), review and label each cluster (pick some points and check for labels), Pick points that are closer to cluster centre and avoid outliers

43.4 Metrics for Clustering

Dataset: Given x and no y;

A good clustering result:

Intra cluster: Within a cluster;

Inter cluster: Between clusters;

Intra cluster should be small and inter cluster should be large: This is how we can effectively measure the performance of the clustering algorithm

Dunn index:

D = min d(i,j) / max d'(k) = minimum inter-cluster distance/ maximum intra-cluster distance

For a good clustering result Dunn index should be large.

43.5 K-Means: Geometric intuition, Centroids

K = number of clusters = hyper parameter;

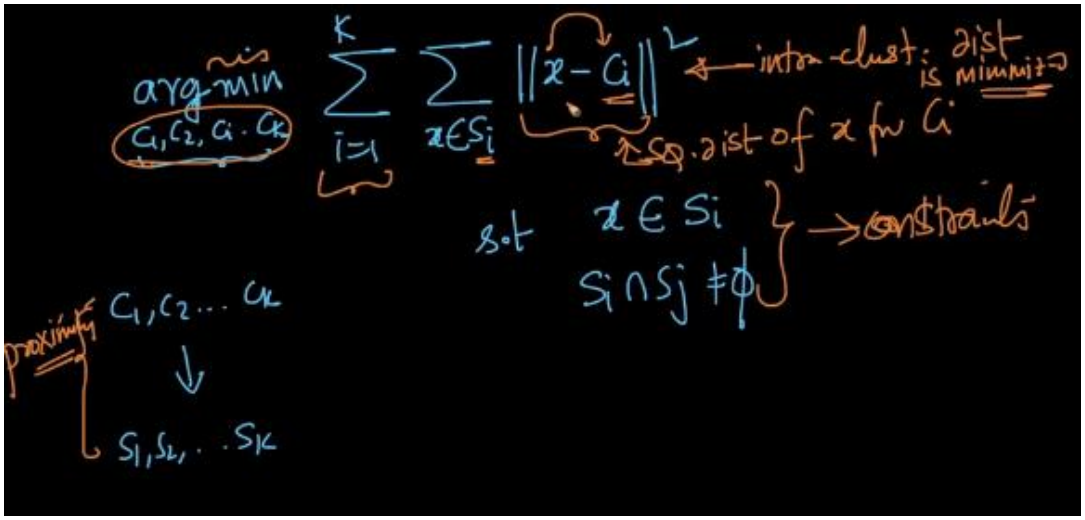
For every cluster K Means assigns a centroid and groups the data points into clusters around the centroid, no point belongs to two clusters and every data point belongs to at least 1 cluster;

Centroid is the geometric mean point of the cluster data points;

K Means is a centroid based clustering scheme;

Data points are assigned to a cluster based on nearness to a centroid;

43.6 K-Means: Mathematical formulations: Objective function



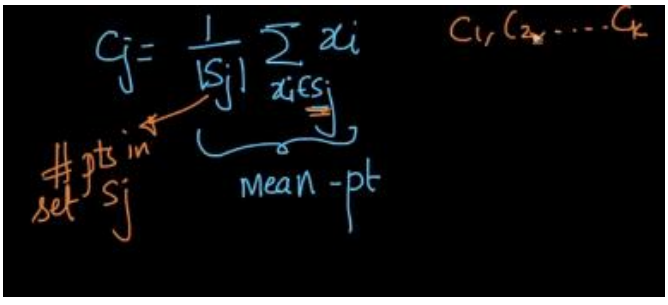
Above formulation has Exponential time complexity: NP hard problem;

An approximation algorithm is used to solve the optimization problem;

43.7 K-Means Algorithm

Lloyd's Algorithm:

1. Initialization: Randomly pick k data points from the Dataset and call them centroids
2. Assignment: For each point in the Dataset, select the nearest centroid through the distance and add this data point to the centroid corresponding cluster
3. Re-compute centroid: Update centroid to the mean of the cluster.

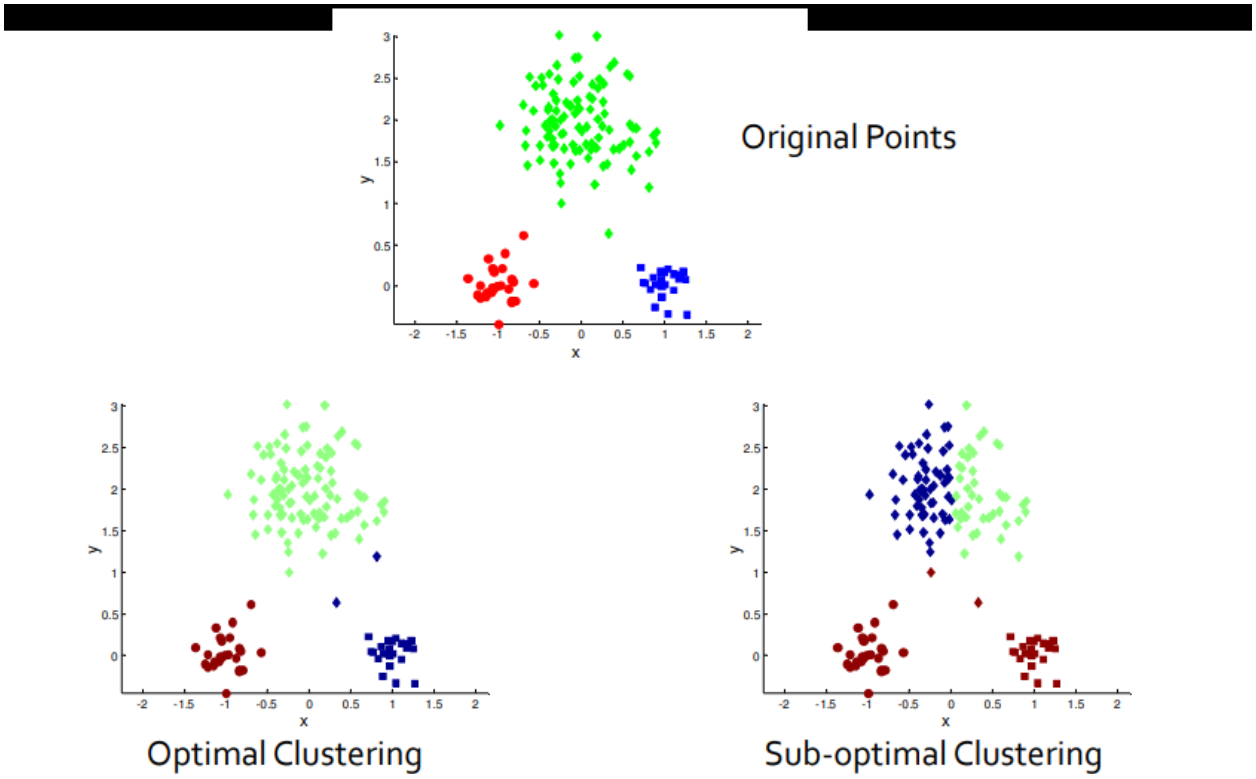


4. Repeat step 2 and step 3 until convergence

43.8 How to initialize: K-Means++

K-Means have initialization sensitivity: the final result changes when the initialization is changed

Link: https://cs.wmich.edu/alfuqaha/summer14/cs6530/lectures/ClusteringAnalysis.pdf



Repeat k means with different initializations: pick best clustering based on smaller intra distances and larger inter cluster distances;

K-Means++: random init is replaced with smart init;

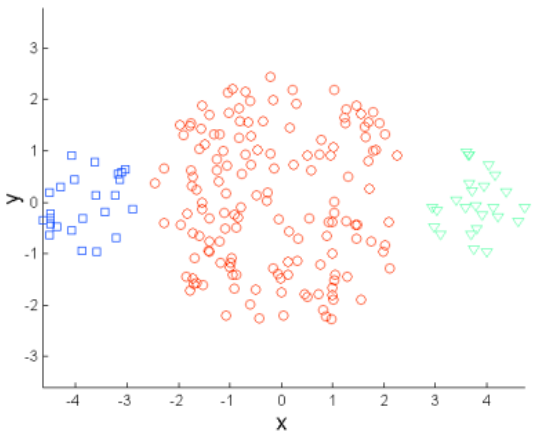
Pick first centroid randomly; pick further centroids with a probability proportional to the distance of the point from the previous nearest centroid;

Probabilistic approach is preferred over deterministic approach where we can pick a point that is far from all centroids, to avoid outliers as K-Means++ gets impacted by outliers and probabilistic approach mitigates this effect;

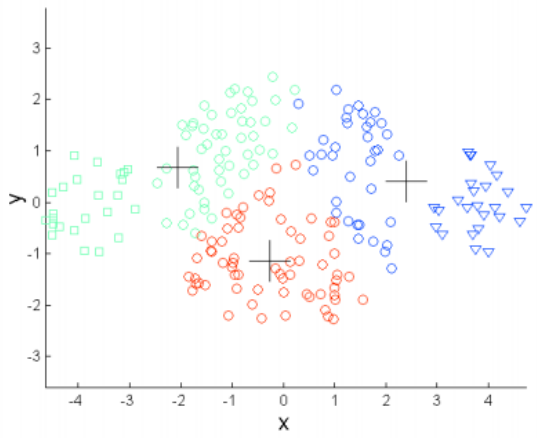
43.9 Failure Cases/ Limitations

Clusters with different sizes, densities and non-globular shapes and presence of outliers

K-Means tries to cluster equal number of data points in each cluster;

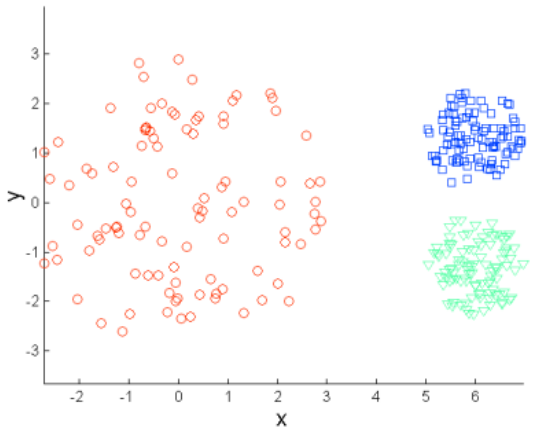


Original Points

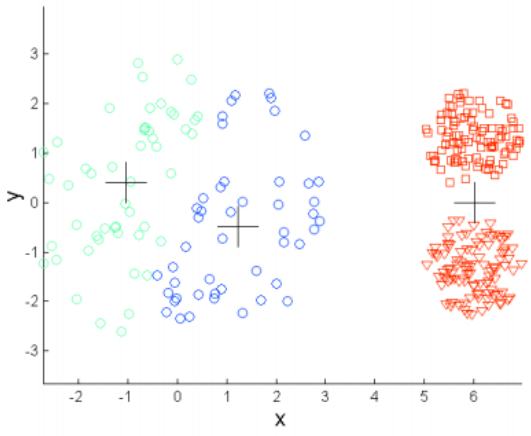


K-means (3 Clusters)

Different densities:



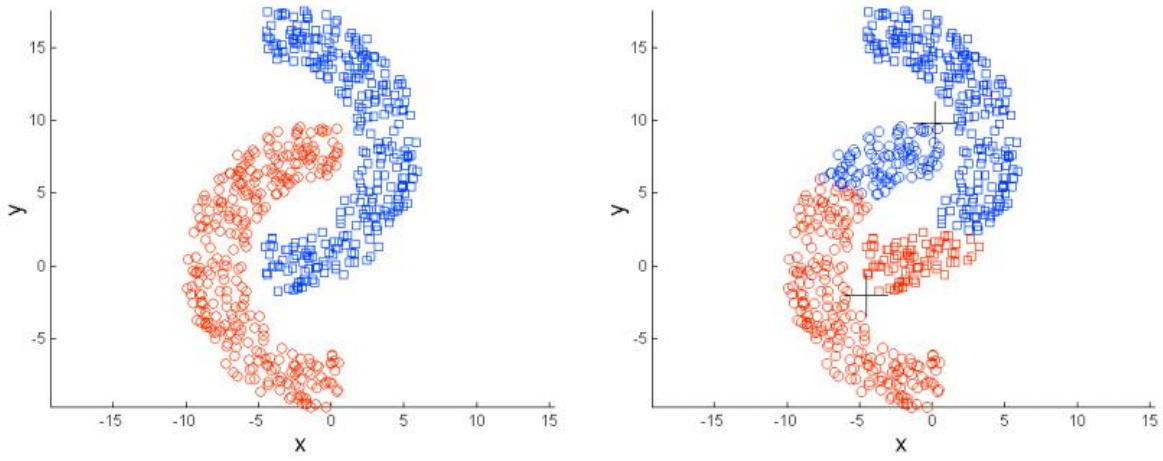
Original Points



K-means (3 Clusters)

K-Means tries to get similar densities across all clusters

Non-Globular shapes:



Original Points

K-means (2 Clusters)

Use larger k to get many clusters and put together different clusters and avoid above problems;

Cluster evaluations are difficult as there is no supervised learning;

43.10 K-Medoids

Centroids may not be interpretable for example bag of words vectorization of text;

Instead of giving centroids computed using means, if we output an actual data point that is just a review will be more interpretable; this review is a Medoid;

Each centroid is a data point = Medoid for interpretation;

Partitioning around Medoids (PAM):

1. Initialization: Similar to K-Means++ (probabilistic approach)
2. Assignment of data points to a cluster (closest Medoid)
3. Update/ recomputed: No mean approach; Swap each Medoid with a non-Medoid point; If loss decreases keep the swap else undo swap; (K-Means formulation, minimizing distances) if swap is successful redo step 2, if a similarity matrix or distance matrix is given, K-Medoids can be easily implemented; K-Medoids is Kernelizable

43.11 Determining the right K

Elbow method: plot loss function vs k; select k with elbow method;

43.12 Code Samples

sklearn.cluster.KMeans

Arguments: n_clusters, n_init

43.13 Time and space complexity

K-Means: O(n*k*d*i) ~ O(nd)

n points, k clusters, d dimensionality, i iterations;

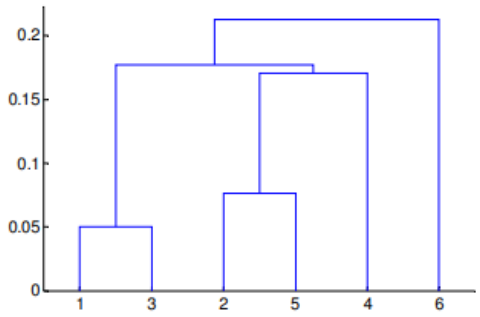
Space: O(nd + kd) ~ O(nd)

Chapter 44: Hierarchical clustering Technique

44.1 Agglomerative & Divisive, Dendrograms

Agglomerative clustering: Takes each data point as a cluster and groups two nearest clusters into one cluster until the number comes to k; Stage wise Agglomeration of clusters;

Divisive starts in the reverse order: Takes all data points into 1 cluster and divides clusters stage be stage; division is a big question; Agglomerative is popular;



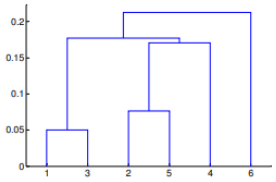
Dendrogram

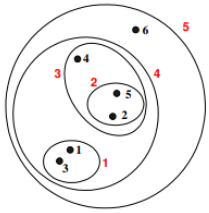
Link: https://cs.wmich.edu/alfuqaha/summer14/cs6530/lectures/ClusteringAnalysis.pdf

44.2 Agglomerative Clustering

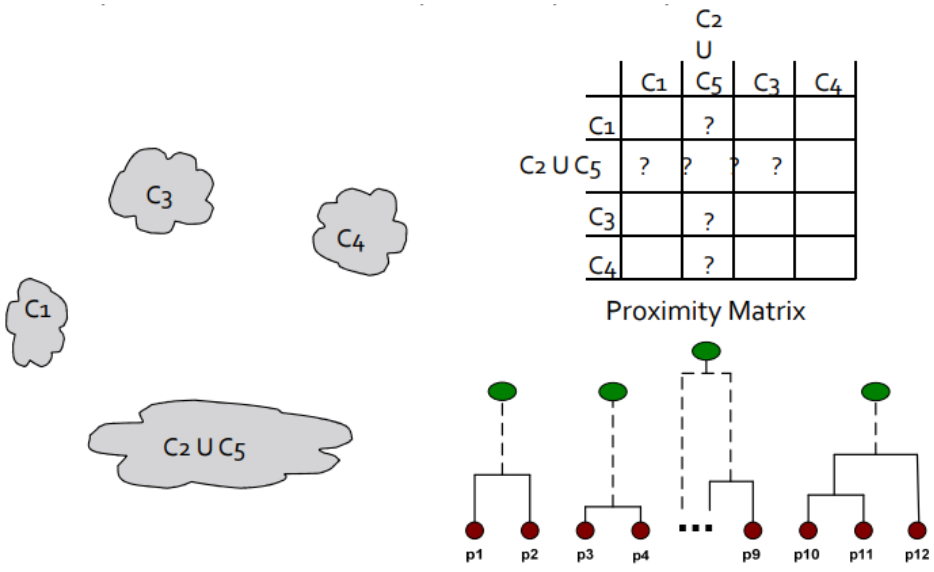
- Two main types of hierarchical clustering
  - Agglomerative:
    - Start with the points as individual clusters
    - At each step, merge the closest pair of clusters until only one cluster (or k clusters) left
  - Divisive:
    - Start with one, all-inclusive cluster
    - At each step, split a cluster until each cluster contains a point (or there are k clusters)
- Traditional hierarchical algorithms use a similarity or distance matrix
  - Merge or split one cluster at a time

- Produces a set of nested clusters organized as a hierarchical tree
- Can be visualized as a dendrogram
  - A tree like diagram that records the sequences of merges or splits





- More popular hierarchical clustering technique
- Basic algorithm is straightforward
  1. Compute the proximity matrix
  2. Let each data point be a cluster
  3. **Repeat**
  4. Merge the two closest clusters
  5. Update the proximity matrix
  6. **Until** only a single cluster remains
- Key operation is the computation of the proximity of two clusters
  - Different approaches to defining the distance between clusters distinguish the different algorithms



44.3 Proximity methods: Advantages and Limitations

Inter cluster Similarity:

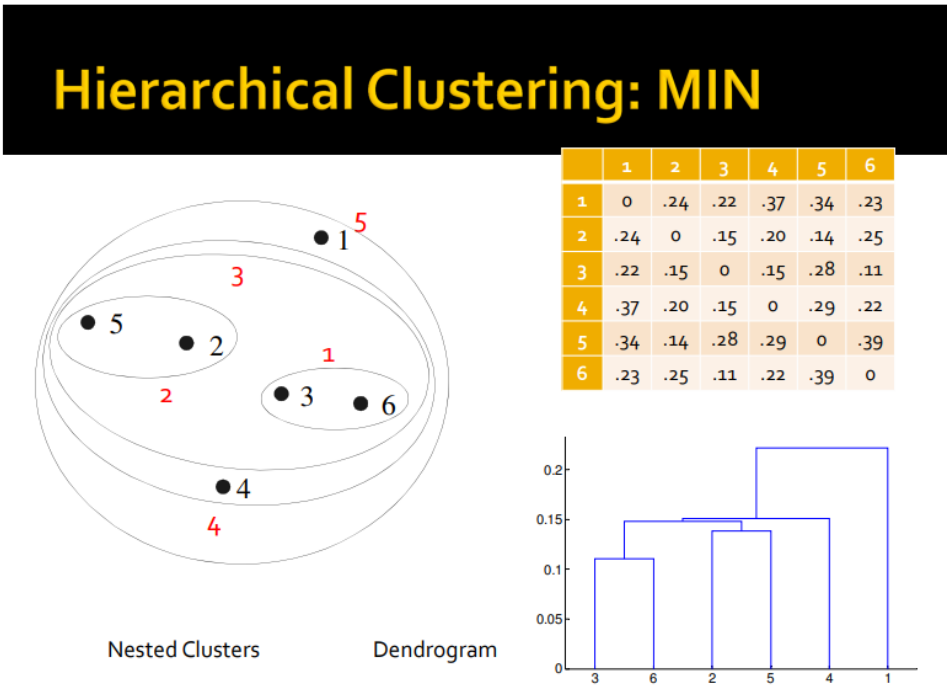
Min: similarity = minimum distance between clusters

Max: maximum distance between clusters

Group Average: Summ(sim(pairs))/ (Multiplication of number of points in all clusters)

Distance between centroids;

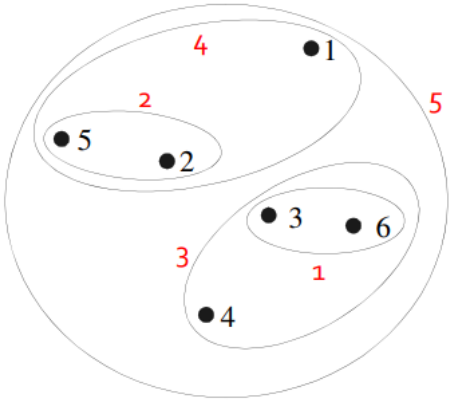
Due to similarity presence in the calculations: Min, Max, Avg can be kernelized



	1	2	3	4	5	6
1	0	.24	.22	.37	.34	.23
2	.24	0	.15	.20	.14	.25
3	.22	.15	0	.15	.28	.11
4	.37	.20	.15	0	.29	.22
5	.34	.14	.28	.29	0	.39
6	.23	.25	.11	.22	.39	0

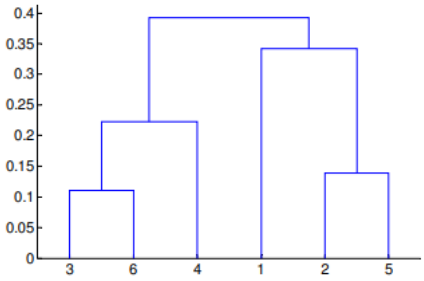
+ Can handle non-globular distributions and - is sensitive to noise and outliers;

Hierarchical Clustering: MAX



Nested Clusters

	1	2	3	4	5	6
1	0	.24	.22	.37	.34	.23
2	.24	0	.15	.20	.14	.25
3	.22	.15	0	.15	.28	.11
4	.37	.20	.15	0	.29	.22
5	.34	.14	.28	.29	0	.39
6	.23	.25	.11	.22	.39	0



Dendrogram

+ Less prone to outliers and noise

- Tends to break large clusters, biased towards globular clusters

Cluster Similarity: Group Average

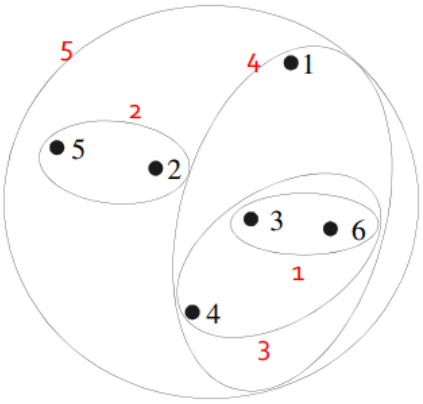
- Proximity of two clusters is the average of pairwise proximity between points in the two clusters.

$$\text{proximity}(\text{Cluster}_i, \text{Cluster}_j) = \frac{\sum_{\substack{p_i \in \text{Cluster}_i \\ p_j \in \text{Cluster}_j}} \text{proximity}(p_i, p_j)}{|\text{Cluster}_i| * |\text{Cluster}_j|}$$

- Need to use average connectivity for scalability since total proximity favors large clusters

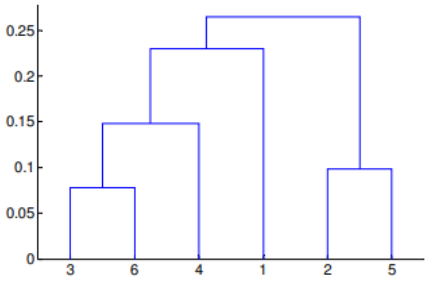
	1	2	3	4	5	6
1	0	.24	.22	.37	.34	.23
2	.24	0	.15	.20	.14	.25
3	.22	.15	0	.15	.28	.11
4	.37	.20	.15	0	.29	.22
5	.34	.14	.28	.29	0	.39
6	.23	.25	.11	.22	.39	0

Hierarchical Clustering: Group Average



Nested Clusters

	1	2	3	4	5	6
1	0	.24	.22	.37	.34	.23
2	.24	0	.15	.20	.14	.25
3	.22	.15	0	.15	.28	.11
4	.37	.20	.15	0	.29	.22
5	.34	.14	.28	.29	0	.39
6	.23	.25	.11	.22	.39	0



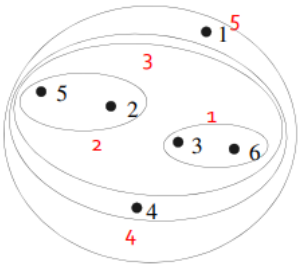
Dendrogram

+ Less prone to outliers and noise

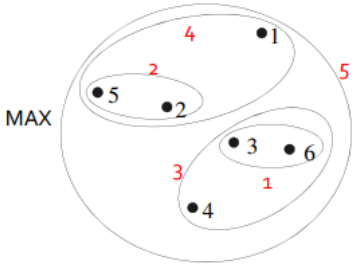
- Biased towards globular clusters

Ward's Method: squared distances are taken; everything else is similar to Group Avg

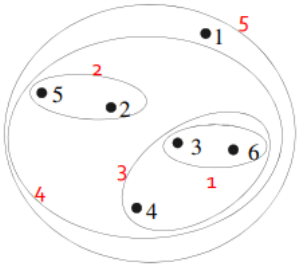
Hierarchical Clustering: Comparison



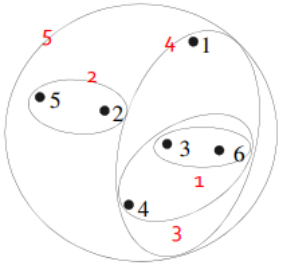
MIN



MAX



Group Average



Ward's Method

44.4 Time and Space Complexity

Space: $O(n^2)$ n number of data points

Time: $O(n^3)$

44.5 Limitations of Hierarchical Clustering

1. No mathematical objective function we are solving in Hierarchical Clustering

2. Computational Complexity (Time and Space)

3. Once a technique is applied it cannot be undone

44.6 Code Sample

sklearn.cluster.AgglomerativeClustering

Hyperparameters: n_clusters, linkage

45.1 Density based clustering

K-Means: Centroid based

Hierarchical: Agglomerative

DBSCAN: Dense regions: Clusters, Sparse: Noise: Separate dataset into dense regions from sparse regions

45.2 MinPts and Eps: Density

Density at point P: Number of points in radius eps around p

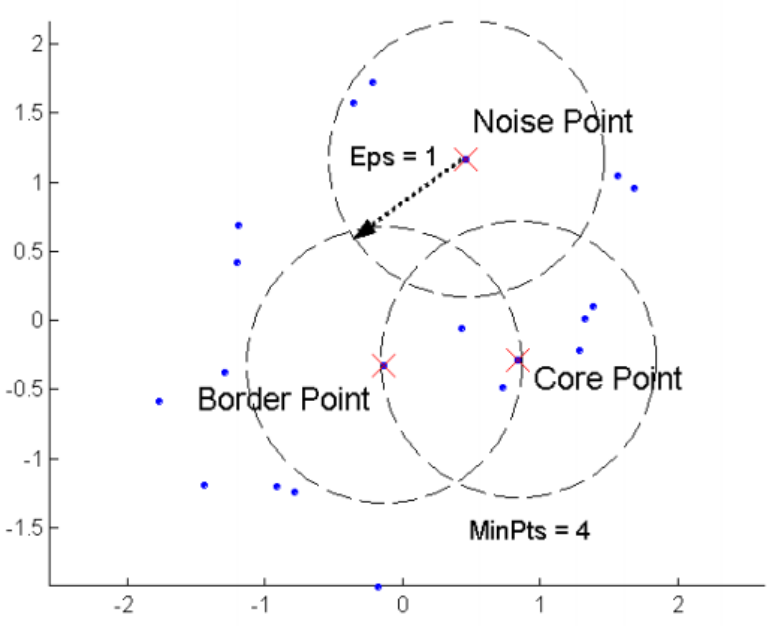
Dense region: Region which has minimum number of points MinPts in radius eps

45.3 Core, Border and Noise points

Core point: if point has >= Min pts in eps radius

Border point: point having <= Min pts in eps radius and belongs to a neighborhood of core point

Noise point: None of above



45.4 Density edge and Density connected points

Density edge: Connection between two core points which are in a radius <= eps

Density connected points: between two core points if there exists all density edges exist in the path between the core points

45.5 DBSCAN Algorithm

1. For each data point: Label each point as core point, border pt or a noise pt (Given Minpts and Eps); This is done using Range Query (using kd-tree or kNN)
2. Remove all noise points as they do not belong to any cluster
3. For each core pt p not assigned to a cluster:

   Create a new cluster with this core pt p

   Add all pts that are density connected to p into this new cluster
4. For each border pts: assign it to the nearest cluster

Important operation: RangeQuery: returns all data points that are in eps radius

45.6 Hyper Parameters: MinPts and Eps

MinPts: >= d + 1, typically 2*d; for d dimensionality, larger Min pts takes care of noise

Or ask an expert for Min Pts value

Eps: For each data point calculate distance from kth nearest point where k = Min pts Sort these distances and plot to get an elbow point

45.7 Advantages and Limitations of DBSCAN

DBSCAN: resistant to noise, can handle different sizes of cluster, does not require n_clusters to be specified;

DBSCAN has trouble with: Varying densities and high dimensionality data, sensitive to hyper parameters: depends on distance measure which causes curse of dimensionality

45.8 Time and Space Complexity

Time: O(n log n)

Space: O(n)

45.9 Code Samples

sklearn.clister.DBSCAN()

45.10 Revision Questions

1. What is K-means? How can you select K for K-means?
2. How is KNN different from k-means clustering?
3. Explain about Hierarchical clustering?
4. Limitations of Hierarchical clustering?
5. Time complexity of Hierarchical clustering?
6. Explain about DBSCAN?
7. Advantages and Limitations of DBSCAN?

46.1 Problem formulation: Movie Reviews

Recommend relevant items to a user based on historical data of the item and the user;

Given dataset A:

Each row is for user and each column is for item

Cell values can be ratings or the usage of the item by the user

Cell values which do not have values are left as nan rather than replacing with 0

Matrix A is very sparse as each user can use a small set of items;

Sparsity = # empty cells/ # total cells; Density = 1 - sparsity

Given a user and his history of item usage, recommend a new item that he will most likely use

Convert the problem into a regression or classification

Task: Feature engineering for given dataset A (user – item pairs)

Converting the problem into a matrix completion problem:

Fill empty cells in the sparse matrix with non empty cell values;

46.2 Content based vs Collaborative filtering

Collaborative Filtering:

U1:- M1, M2, M3

U2:- M1, M3, M4

U3:- M1

As M1 is common movie watched by user1, user2 and user3, and movie M3 is the common movie watched by user1 and user2, movie M3 can be recommended to U3

Idea: Users who agreed in the past tend to agree in the future (assumption for collaborative filtering)

Content based: uses rating information or the usage matrix values as target variable;

Uses representation of the item and the user (features), such as his preference, gender, location, item type, movie type, movie title, movie cast, etc

46.3 Similarity based Algorithms

Item-item based similarity or user-user similarity;

User-User similarity:

Given matrix with every row of the matrix is a user vector and every column is an item;

Build user-user similarity matrix; Say we have U1, U2, and U3, who are most similar to U4, we can recommend items that U4 has not used yet and that U1, U2 and U3 have used already and recommend these items to U4

1. Build user vector based on ratings or item usage
2. Build user – user similarity matrix using cosine similarity: $U_i^T U_j/(||U_i||*||U_j||)$
3. Find similar users and the items they have in common
4. Find items that are not watched by the user
5. Recommend the un common item

Limitation: User's preference change over time: thus user-user similarity approach does not work well (when the change is very frequent)

Item-item similarity:

Each item is a vector; and a similarity matrix is build using similarity between items;

Ratings on a given item do not change significantly after the initial period;

If we have more users than items and when item ratings do not change much over time after initial period, item – item similarity matrix is preferred

46.4 Matrix Factorization: PCA, SVD

Also called Matrix decomposition:

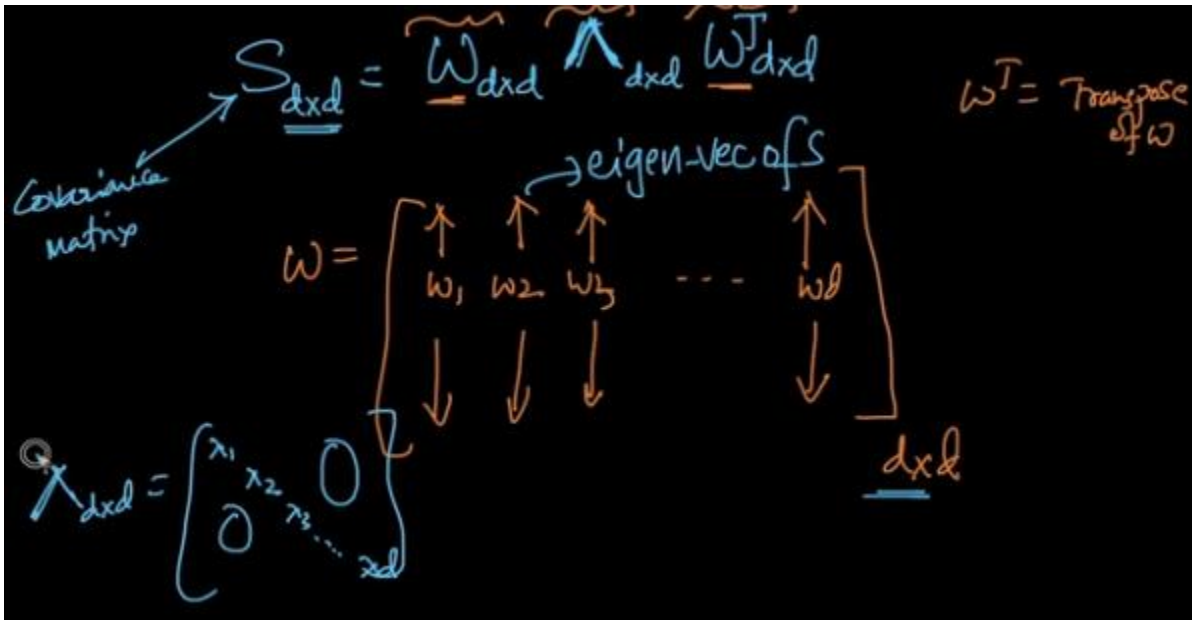
Decomposing a matrix into a product of other matrices:

A = B*C*D = P*Q

Principal Component Analysis: Dimensionality Reduction: Can be interpreted as an example of Matrix factorization

X_{nxd} → Data Matrix; $S_{dxd} = X^TX$ = co-variance matrix

Eigen values and Eigen vectors of S can be determined and when reducing d dimension to d' dimension we take top eigen values and corresponding eigen vectors to project data points from d to d';



- Eigen decomposition of Covariance matrix

Singular Value Decomposition: Matrix Factorization related to PCA

PCA: Eigen value decomposition of covariance matrix;

SVD: Any rectangular matrix (non square matrix)

$X_{nxd} = U \Sigma V^T$ = (nxn) * (nxd) * (dxd)

Σ = diagonal matrix of singular values of X (related to eigen values)

Singular value $s_i = ((n-1)\ \lambda_i)^2$

U: contains Left singular vectors

U^T: contains Right singular vectors

Σ: contains Singular Vectors

46.5 Matrix Factorization: NMF

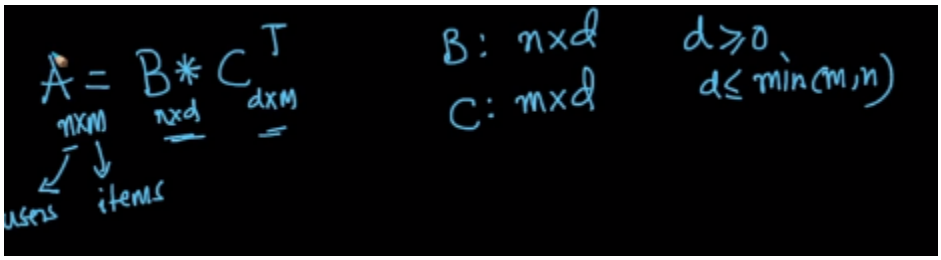
Non-negative Matrix Factorization:

$A_{nxm} = B_{nxd}\ (C^T)_{dxm}$

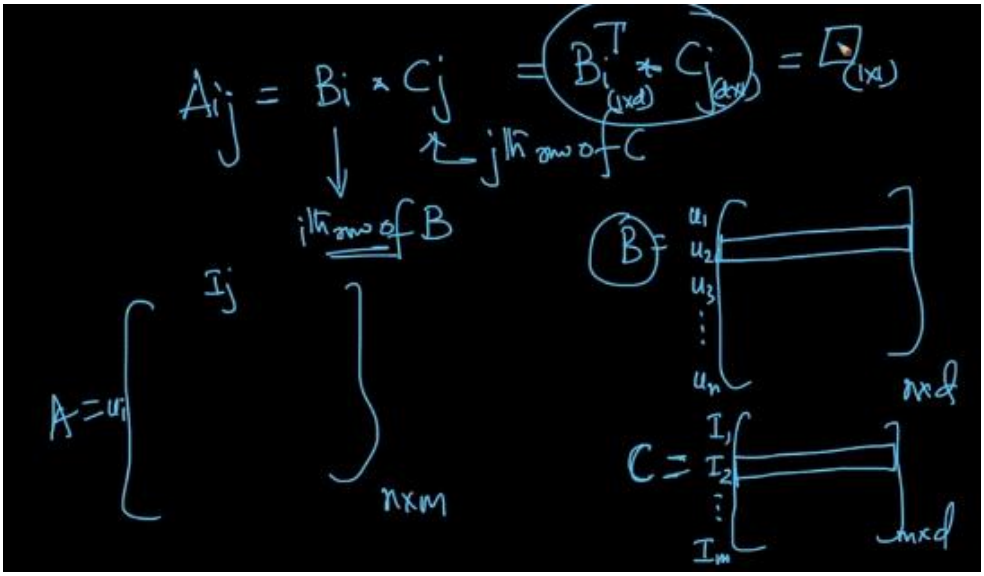
Such that, $B_{ij} >= 0$ and $C_{ij} >= 0$

46.6 Matrix Factorization for Collaborative filtering

A_{ij} = rating on $Item_j$ by $User_i$



Let decomposition be possible:

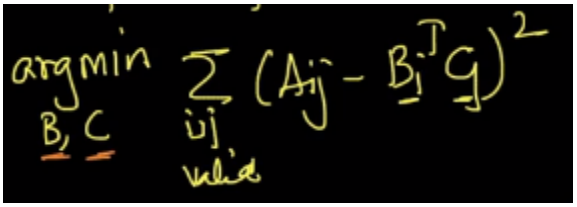


Find B & C:

$$\underset{B,C}{\text{argmin}} \sum_{ij \text{ valid}} (A_{ij} - B_i^T C_j)^2$$

Minimizing square loss: Actual Matrix and its Matrix Factorization

1. Solve optimization problem using SGD:



Used non empty A values to get B and C;

2. Compute B and C
3. Matrix completion: Fill the empty cells with B and C

46.7 Matrix Factorization for feature engineering

Matrix A of user item ratings: Through Matrix factorization we get B and C

d-dimensional representation of user and item: using A values;

Row vector of B matrix above can be used as $user_i$ vectorization and from C we can have $item_j$ vectorization;

The d-dim representation arrived at using Matrox Factorization: if two users are similar then the distance between vectors will be small, similarly for items;

Matrix Factorization can be used for:

Word vectors

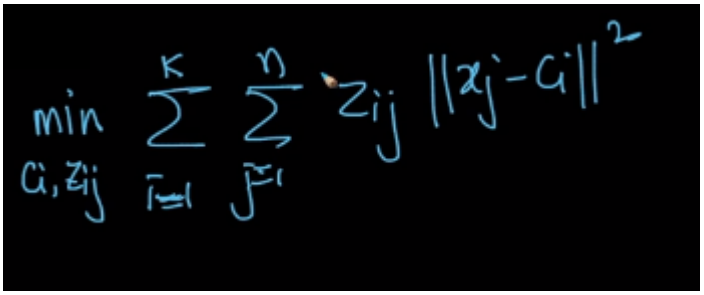
Eigen faces (face recognition) (through feature engineering)

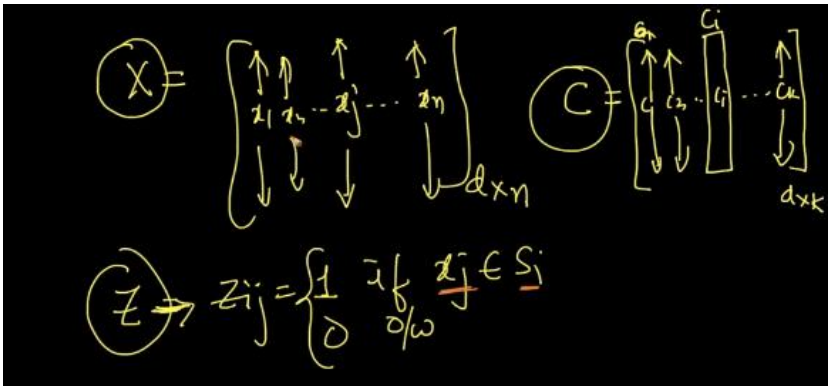
46.8 Clustering as Matrix Factorization

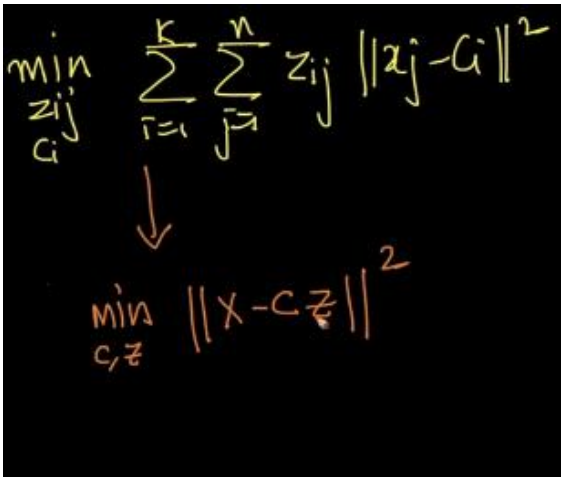
K-Means optimization: D = $\{x_i\}$

> Find k- cluster centroids and corresponding sets of data points; Such that every data point belongs to only one set and the distance from the data points to the cluster centroid is minimum;

Define a matrix Z such that $Z_{ij} = 1$ if x_j belongs to Set S_i else 0; The Matrix Z is sparse and can be said to be an assignment problem;







If X is decomposed into C and Z through Matrix Factorization:

such that $Z_{ij} = 0$ or 1

and sum of all elements of $Z_j = 1$

K-Means is Matrix Factorization + Col constraints + 0,1 values

46.9 Hyperparameter tuning

$A_{nxm} = B * C^T$ by Matrix Factorization

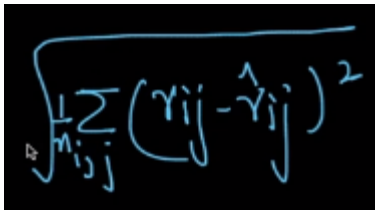
d-dimensionality is a Hyperparameter;

1. Problem specific
2. Systematic way:: Optimization: min $(A - BC)^2$
   Error plot with d dimensionality is generated and an elbow point is selected

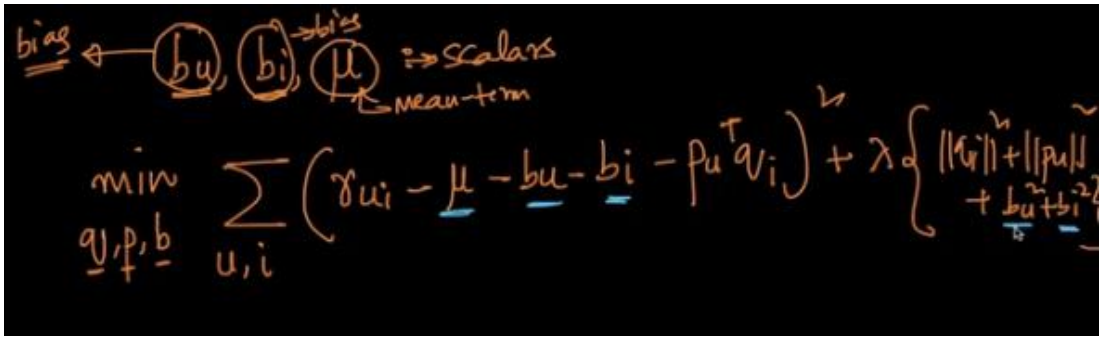
46.10 Matrix Factorization for recommender systems: Netflix Prize Solution

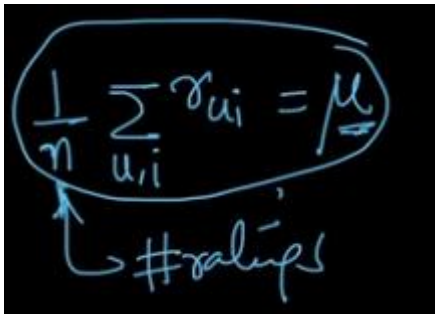
In 2009: Streams Video on demand;

Given user – ratings and a loss metric (RMSE)



RS → MF became popular only after Netflix prize;





b_u = bias due to user, b_i = bias for item

Ratings for users and items are time dependent;

Link: https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf

46.11 Cold start problem

If there is a new user joins the system or a new item that is added, then there is no ratings data;

Recommend top items based on meta-data such as geo location, browser, and device;

We can make content based recommendation;

46.12 Word Vectors as Matrix Factorization

Word2Vec: Inspired by Neural Networks, ex: LSA< PLSA< LDA< GLOVE can be interpreted as Matrix Factorizations

GLOVE: related to W2V

SVD related to PCA for W2V

1. Co-occurrence matrix:

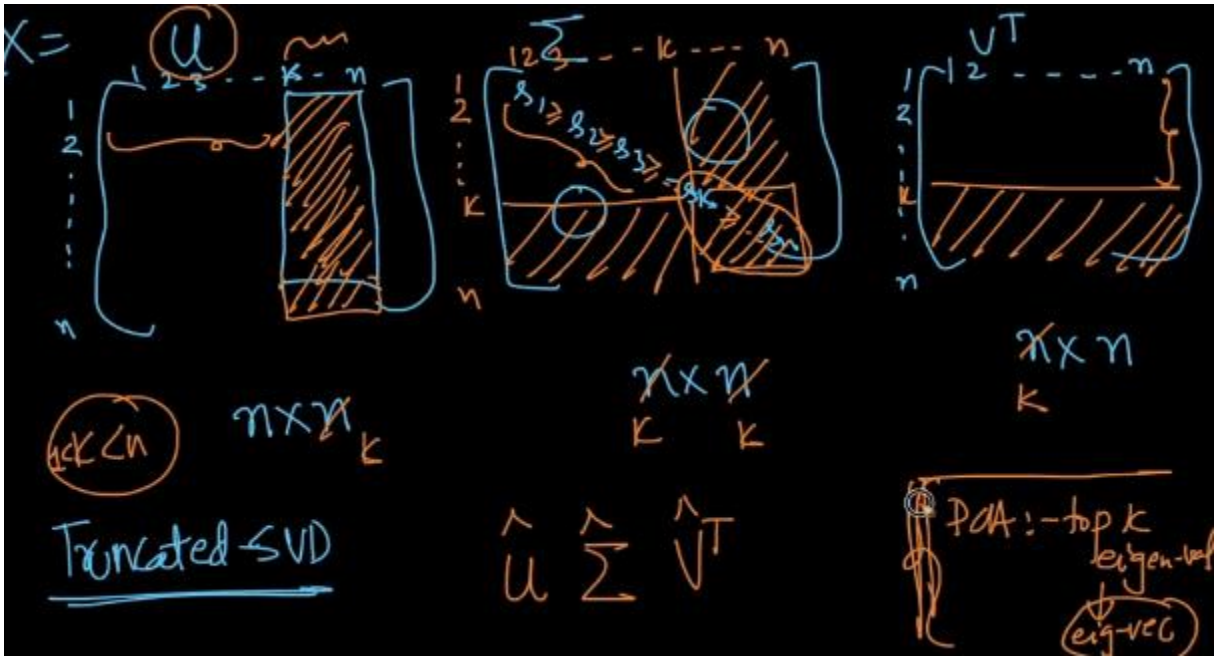
   D = {text documents}

   Compute X matrix:

   X_{ij} = # of time w_j occurs in the context of W_i;

2. $X_{nxn} = U \Sigma V^T$ (nxn each): Matrix Factorization (SVD)

   Using Truncated SVD by discarding (n-k) columns



Truncated SVD: top k singular values, top k left singular vectors and top k right singular vectors: discarding Eigen values with least information; k is the Hyperparameter;

Using co-occurrence matrix and applying truncated SVD over the matrix we will get U matrix. From this U matrix which is of (nxk) shape, we have each row as vector representation of each word of k dimensionality

Instead of taking all words we can have top words that have good importance;

46.13 Eigen-Faces

Word vectors → MF (SVD) → Co-occurrence matrix)

Image data: Eigen faces for face recognition (PCA) to get feature vectors; (CNN replaced all techniques for image tasks)

Link: https://bugra.github.io/work/notes/2014-11-16/an-introduction-to-unsupervised-learning-scikit-learn/

Link: http://scikit-learn.org/stable/auto_examples/decomposition/plot_faces_decomposition.html#sphx-glr-auto-examples-decomposition-plot-faces-decomposition-py

Matrix construction with stacking images row wise where each row in the matrix contains image data which is flattened into a single vector;

From this matrix: Co-variance matrix is computed; dimensionality reduction is applied on this co-variance matrix through Matrix Factorization;

Multiply row wise images stacked matrix with the top k left singular vectors or column wise stacked Eigen vectors of top k eigen values of the co-variance matrix; This multiplication result is the Eigen Faces

Through Eigen values we can compute % of explained variance (ratio of Eigen values) to get good k;

46.14 Code example

sklearn.decomposition.TruncatedSVD()

sklearn.decomposition.NMF()

https://stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca

Elastic Net: min loss + alpha*L1 norm + (1 – alpha) L2 norm

L1 norm induces sparsity

46.15 Revision Questions

1. Explain about Content based and Collaborative Filtering?
2. What is PCA, SVD?
3. What is NMF?
4. How to do MF for Collaborative filtering?
5. How to do MF for feature engineering?
6. Explain relation between Clustering and MF?
7. What is Hyperparameter tuning?
8. Explain about Cold Start problem?
9. How to solve Word Vectors using MF?
10. Explain about Eigen faces?

Chapter 47: Interview Questions on Recommender Systems and Matrix Factorization

47.1 Question & Answers

1. How would you implement a recommendation system for our company's users?(https://www.infoworld.com/article/3241852/machine-learning/how-to-implement-a-recommender-system.html)
2. How would you approach the "Netflix Prize" competition?(Refer http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/)
3. 'People who bought this, also bought…' recommendations seen on amazon is a result of which algorithm? (Please refer Apparel recommendation system case study, Refer:https://measuringu.com/affinity-analysis/)

Chapter 48: Case Study 8: Amazon fashion discovery engine (Content Based recommendation)

--- Case Studies Notebook

Chapter 49: Case Study 9: Netflix Movie Recommendation System (Collaborative based recommendation)

--- Case Studies Notebook

Module 8: Neural Networks, Computer Vision and Deep Learning

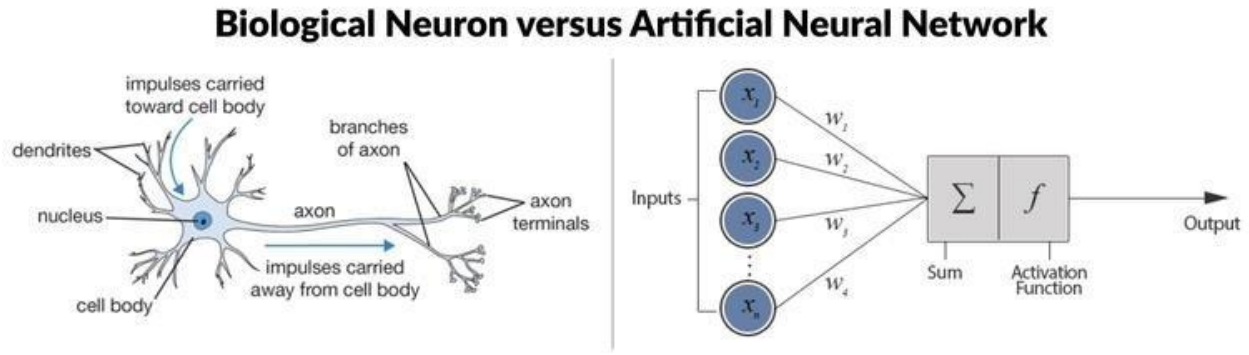
Co-authors: Karthik Kumar Billa, [LinkedIn] Course: Applied AI Course

---

50.1 HISTORY OF NEURAL NETWORKS AND DEEP LEARNING.

- Neural Networks are an important Machine Learning Modelling framework
- First simplest Neural Network Model built was a Perceptron. Built in 1957 by Rosenblatt
- https://en.wikipedia.org/wiki/Perceptron
- With small changes a perceptron can become a logistic regression
- After WW2, in US people were trying to translate messages from Russain to English
- Contributors - Alan Turing(father of modern computing)
    - Curiosity raised these questions:
        - What is intelligence?
        - How should we build it artificially?
- There was a biological inspiration
- Due to work in NeuroScience, a vague understanding of the working of brain has been developed

Biological Neuron vs Artificial Neuron



Biological Neuron Artificial Neuron

A biological Neuron has Nucleus, Dendrites and a cell body. When a Neuron gets electrical signals as inputs, the neuron does computations inside and it sends the processed electrical signals as output maybe to other neurons.

An Artificial Neuron has some inputs that are important, thus weights are included on edges of the connections.

Output = f(W1x1 + W2x2 + W3x3 + ……)

Perceptron is this single neuron which is loosely inspired from the biological neuron and is not an exact replica but is powerful enough to solve many interesting problems.

In biology, a neuron does not exist on itself. It is connected to other neurons. A structure of neurons can be considered for imagination (network). First successful attempt was made in 1986 by a group of mathematicians (Hinton and others). They came up with backpropagation (chain rule around differentiation). A lot of hype has been generated.

Unfortunately around 1996, due to insufficient computational power, insufficient data and lack of algorithmic advancements AI experienced a long winter. This is shortly called AIWinter. Funding for AI got exhausted due to hype. Neural Networks couldn't take off in the 90s. People shifted to SVM, RandomForest and other GBDTs between 1995 to 2009, which were giving solutions to many problems.

Hinton in 2006 released a paper on how to train a Deep Neural Network. Before this NN was limited to a small number of layers. As the number of hidden layers increased backpropagation failed.

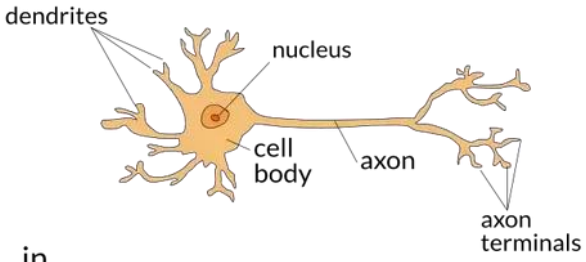
DNN took developments with a competition on ImageNet Dataset. The task was to identify objects in images. DNN has performed very well by a large margin on this dataset compared to other classical ML algorithms.

Noticeable Applications: Siri, Cortana, Alexa, Google Assistant, Self Driving Cars, Health care purposes

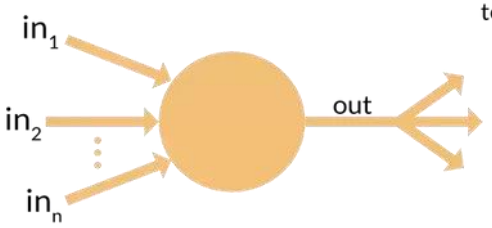
The development is also driven by the availability if Data, computational power and new algorithms.

50.2 HOW BIOLOGICAL NEURONS WORK?

- Biological Neuron:



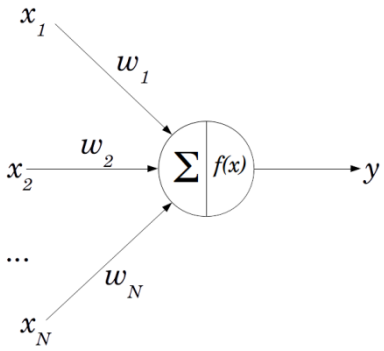
Biological Neuron



Simplified view of Neuron

Lof of internal chemical activity takes place in a biological neuron, but we can understand its working with a simple structure (as above). We have biochemistry generated electrical signals.

Each neuron gets connected to a bunch of other neurons. Some dendrites are thicker. This leads to more weight for that input. A neuron is activated or fired if there is enough input.



f(x) is the activation function that transforms input to a desired output.

Output y = f(sum(Wi*xi)) = f(W1x1 + W2x2 + W3x3 + …)

We are applying activation function on summation of weighted inputs.

Output dendrites will also have different weights.

The main purpose of the activation function is to add non-linearity in the network so that the model can learn difficult objective functions.

In ensembles we are training different models and we are combining them on specific conditions. In NN, all neurons are learning at the same time based on the loss function we have.

Generally we use a nonlinear activation function. For regression problems we use a linear activation function in the last layer.

50.3 GROWTH OF BIOLOGICAL NEURAL NETWORKS

Let us consider the growth of a neural network in the human brain. At birth, there are far few connections between neurons. Missing connections can be thought of having weights equal to 0. At age 6, the network gets dense, i.e. weights get trained. At age 60, weights or connections of some edges disappear or become thin, this process is termed as Neural degeneration.

By age 6, humans learn: language, object recognition, sentence formation, speech, etc. (massive amounts of learning). Biological learning is basically connecting neurons with edges. New connections are formed based on data (not random).

50.4 DIAGRAMMATIC REPRESENTATION: LOGISTIC REGRESSION AND PERCEPTRON

Logistic regression: a plane separating positive points from negative points.

LR: given x_i predict y_i

$$y\_i\_hat = sigmoid(W.T*x\_i + b)$$

Given a dataset: D = {x_i, y_i}, we find W and b while training LR

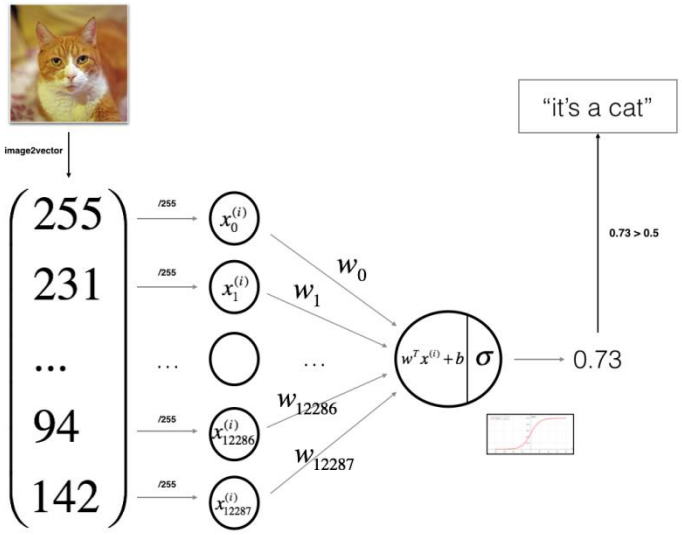
Let x_i E R(d) → W E R(d)

X_i = (x_i1, x_i2, ……, x_id)

W = [W1, W2, ….., Wd]

$$y\_i\_hat = sigmoid\ (summ(j = 1\ to\ d)(W\_j * x\_ij + b))$$

$$= f(summ(j = 1\ to\ d)(W\_j * x\_ij + b))$$



Sigmoid (logistic) activation function: Representation of Logistic Regression with a simple neuron.

If the activation function is sigmoid, a simple perceptron becomes an LR model.

Given dataset of {x_i, y_i}

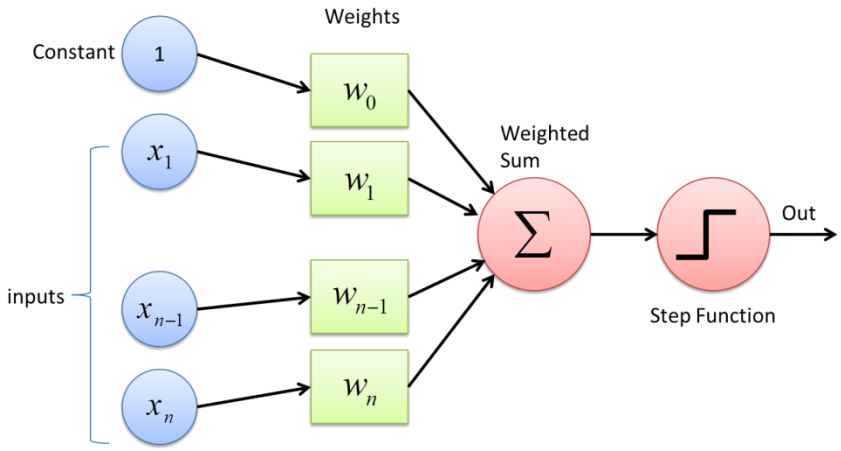
Task is to find W and b: trained by SGD

Training a neural network → finding weights on edges.

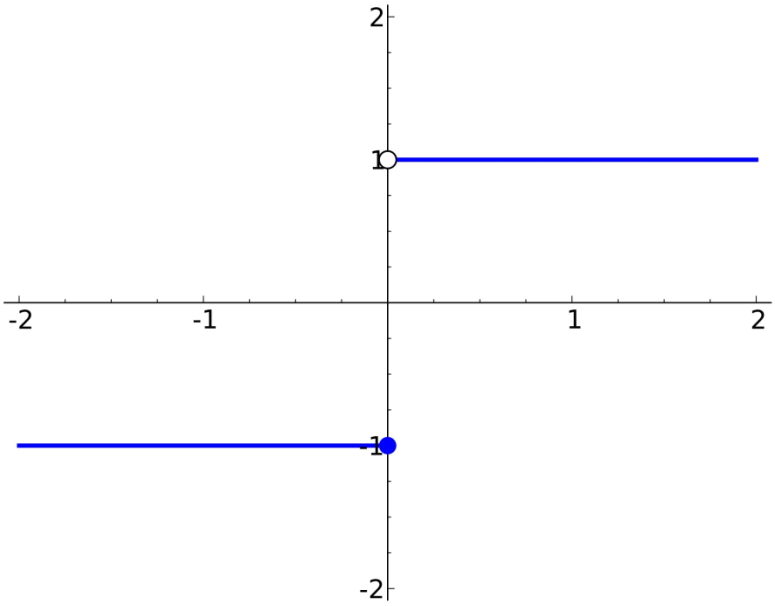
- **Perceptron** - 1957 - Rosenblatt

(difference between logistic regression and perceptron is in the activation function)

For perceptron: f(x) =[1 if W.T * x + b >0; 0 otherwise]



Perceptron



Perceptron activation function

A perceptron can also be understood as a linear function trying to find a line that separates two classes of datapoint.

In logistic regression we have squashing through sigmoid. In perceptron we have no squashing. We just have a step function as an activation function.

LR, perceptron → Simple Neural network

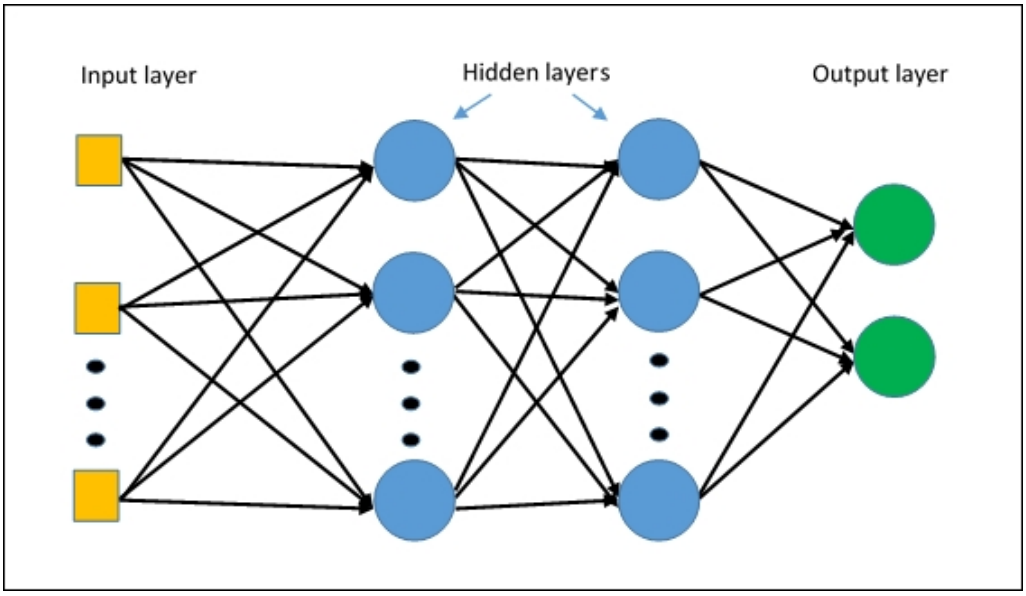
→ difference in activation functions

50.5 MULTI-LAYERED PERCEPTRON (MLP).

Perceptron: a simplified single neuron

Looks alike Logistic Regression

An experiment with connecting multiple neurons was done.

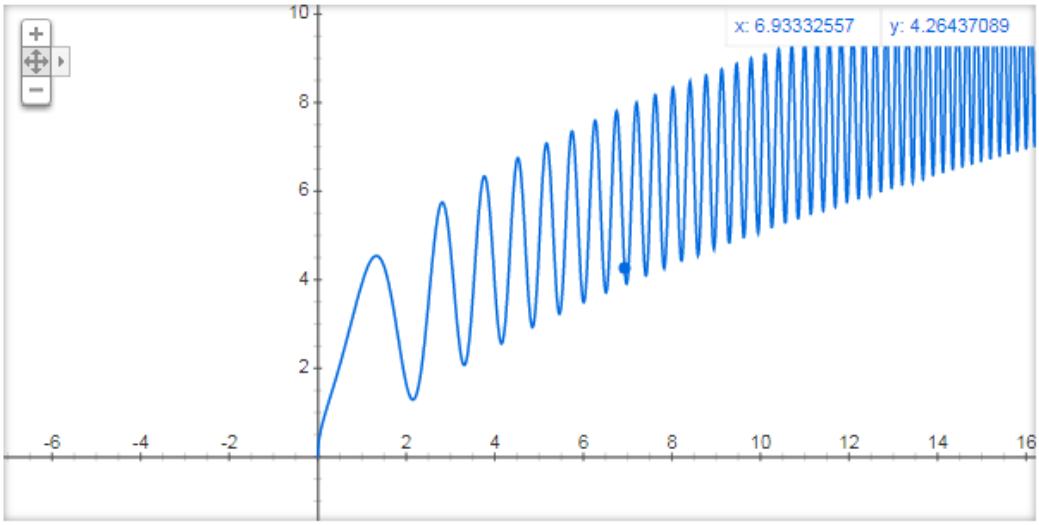


Bunch of single neurons stacked to form a layer and layers are stacked to form a network of neurons.

Q. Why should we care about the Neural network??

- Biological inspiration: connected structure like as in brains, to get true performance (making them work took decades)
- Mathematical argument: Given: D: {x_i, y_i}
    - task : find f where f(x_i) = y_i (belongs to R)
    - Ex: D = {x_i, y_i}
    - Let x_i belong to R(1 dim) and y_i belong to R(1 dim)
    - Let f(x) = 2*sin(x^2)+(5*x)^0.5

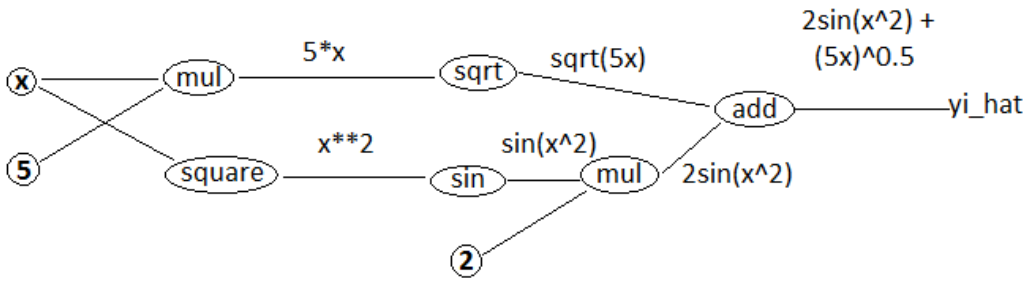
Graph for 2*sin(x^2)+(5*x)^0.5



- Case: with a simple function y = f(x) = x
- We will get a 45 degree line passing through origin
- For 2*sin(x^2)+(5*x)^0.5 we require a complex function, a single linear regression is not sufficient

- Understanding why MLPs are powerful:

Let f1 → add(); f2 → square(); f3 → sqrt(); f4 → sin(); f5 → mul()



f (x) = 2sin(x^2) + (5x)^0.5 = f1(2sin(x^2), (5x)^0.5) = f1(f5(2,sin(x^2)), f3(5x))

= f1(f5(2,f4(x^2)), f3(f5(5,x)) = f1(f5(2,f4(f2(x))), f3(f5(5,x)))

This can be thought of as a function of functions. Thus with MLP we can have complex functions to act on x to get y. Having MLPs we can easily overfit, thus regularizers are applied to avoid overfit. MLP is a graphical way of representing functional compositions.

50.6 NOTATION

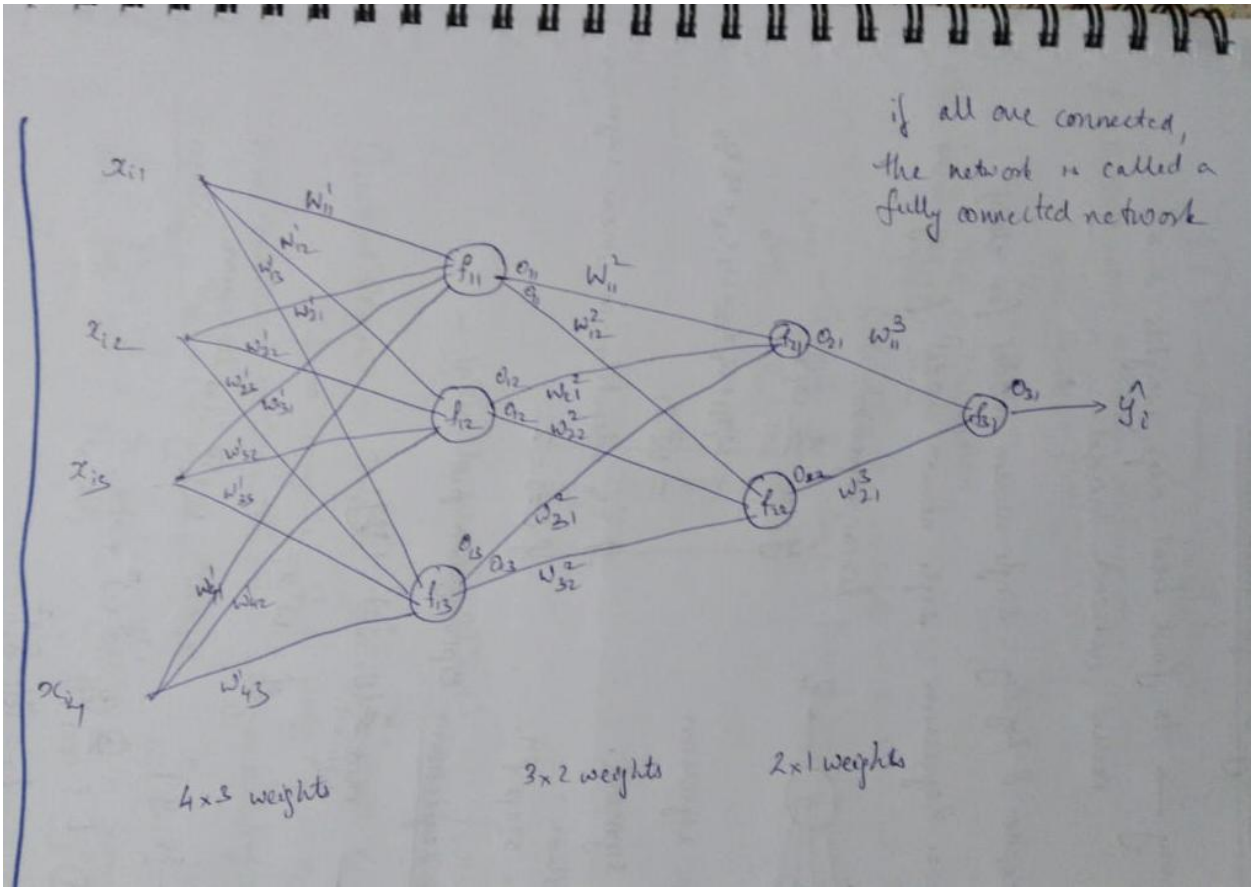
MLPs generate powerful models.

Let D = {x_i, y_i}; x_i belongs to R(4) and the problem is a regression problem

f_ij → function of i layer j neuron

o_ij → output of i layer j neuron

W_k_ij → from i neuron, to j neuron, to k layer



50.7 TRAINING A SINGLE-NEURON MODEL.

Training → to find best edge weights in a neural network model.

Perceptron and Logistic Regression: Single neuron model for classification

Linear Regression: single neuron model for regression

- Linear Regression: y_i_hat = W1*x_i1 + W2*x_i2 + W3*x_i3 + …… = W.T*x_i

  Activation function f is identity in linear regression

  f(z) = z

  For logistic regression f = sigmoid, for perceptron f = step function

Linear regression: Optimization problem:

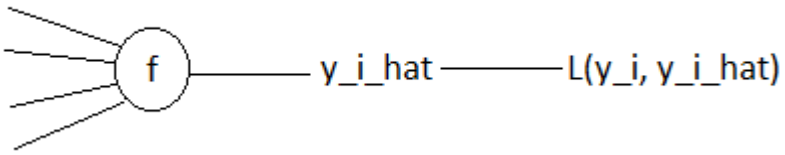
min (Wi) summ(y_i - y_i_hat) ^2 + regularization

Y_i_hat W.T * x_i

D = {x_i, y_i} (i = 1 to n); x_i E R(4), y_i E R : regression

1. Define Loss function: L = summ(y_i - y_i_hat) ^2 + regularization

   L-i = (y_i - y_i_hat)^2



2. Optimization problem definition:

   min (W_i) summ(y_i - f(W.T * x_i))^2 + reg

3. Solve the optimization
   - Usin SGD
   - Initialize the weights → random initialize and initialize eta
   - Key step - **Compute derivative of loss function w.r.t weights**

$$\nabla_w L = \begin{bmatrix} \frac{\partial L}{\partial W_1} \\ \frac{\partial L}{\partial W_2} \\ \frac{\partial L}{\partial W_3} \\ \frac{\partial L}{\partial W_4} \end{bmatrix}$$

- W_new = W_old - $\eta(\nabla_w L)_{W\_old}$

Gradient Descent → Compute $\nabla_W L$ using all x_i's and y_i's

Stochastic Gradient Descent → Compute $\nabla_W L$ using all x_i and y_i

Mini-batch SGD → Compute $\nabla_W L$ using a batch of x_i's and y_i's

Computing $\nabla_W L$: Using chain rule:

Using path from W1 to L:

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial f} * \frac{\partial f}{\partial W_1}$$

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial f} * \frac{\partial f}{\partial W_1}$$

$$L = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$= \sum_{i=1}^{n} (y_i - f(W^T x_i))^2$$

$$\frac{\partial L}{\partial f} = -\sum_{i=1}^{n} 2 * (y_i - f(W^T x_i))$$

$$\frac{\partial f}{\partial W_1} = x_{i1}$$

$$\frac{\partial L_i}{\partial W_1} = -2x_{i1} * (y_i - f(W^T x_i))$$

$$\frac{\partial L}{\partial W_1} = -\sum_{i=1}^{n} 2x_{i1} * (y_i - f(W^T x_i))$$

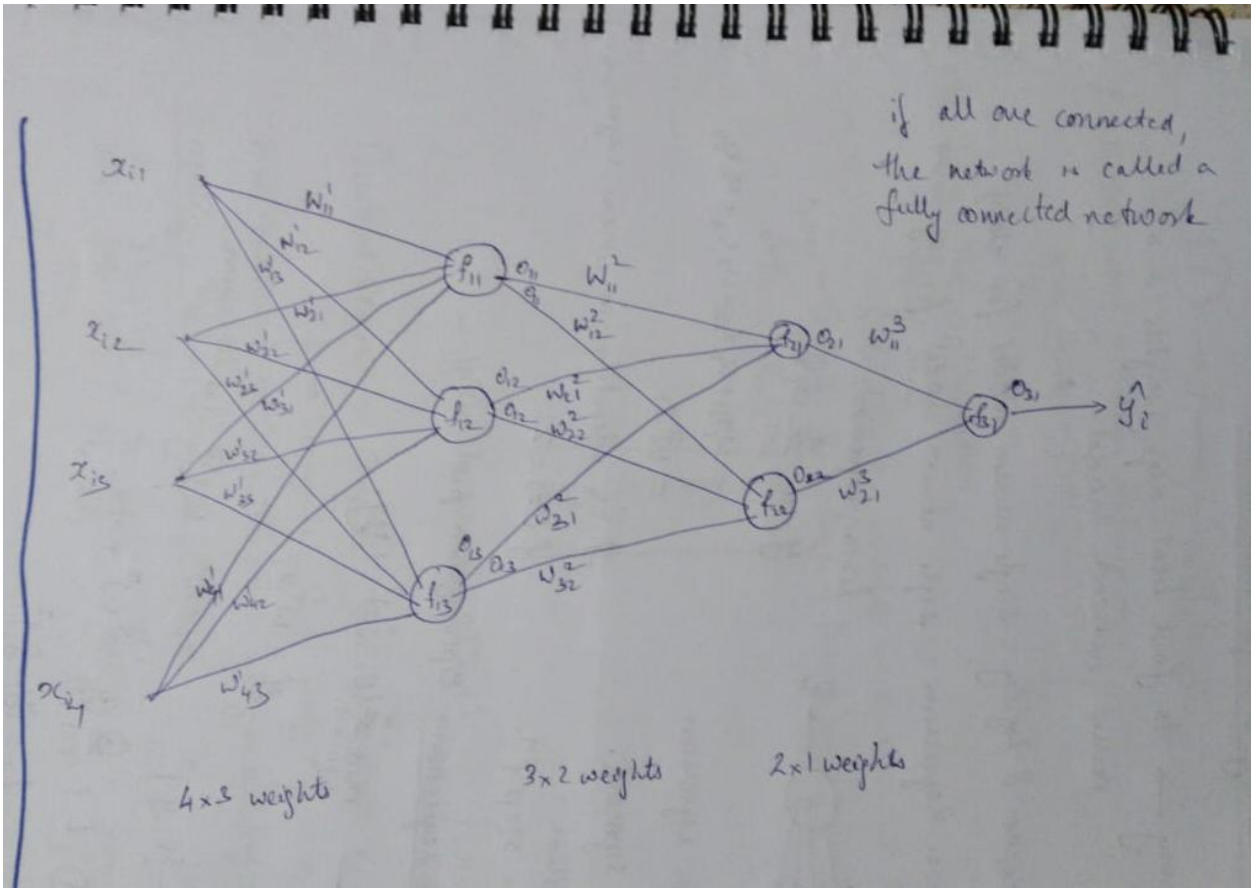
Similarly, compute $\frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial W_3}, \ldots$

50.8 TRAINING AN MLP: CHAIN RULE

- Backpropagation used for training

D = {x_i, y_i}

x_i belongs to R(4), y_i belongs to R, square loss L(y_i, y_i_hat)



$$\frac{\partial L}{\partial {W_{11}}^1} = \frac{\partial L}{\partial O_{31}} * \frac{\partial O_{31}}{\partial {W_{11}}^1} = \frac{\partial L}{\partial O_{31}} * \left(\frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial O_{11}} + \frac{\partial O_{31}}{\partial O_{22}} * \frac{\partial O_{22}}{\partial O_{11}}\right) * \frac{\partial O_{11}}{\partial {W_{11}}^1}$$

(Consider path of flow of feature)

$$\mathrm{L} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \mathrm{reg}$$

1. Define loss function:
2. Optimization: min L (find weights that minimize the loss function)
3. SGD →
    a. Initialize WK_ij randomly

$$W_{ij}^{k}{}_{new} = W_{ij}^{k}{}_{old} - \eta * \left(\frac{\partial L}{\partial W_{ij}^{k}}\right)_{W_{ij}^{k}{}_{old}}$$

    b. (η = Learning rate)
    c. Perform Updates (step 2) upto convergence

    Until: W_new ~ W_old

In the notation (figure), we have W^3_{11} impacting O_{31} and O_{31} impacting L

$$\frac{\partial L}{\partial {W_{11}}^3} = \frac{\partial L}{\partial O_{31}} * \frac{\partial O_{31}}{\partial {W_{11}}^3}$$

- $$\frac{\partial L}{\partial {W_{21}}^3} = \frac{\partial L}{\partial O_{31}} * \frac{\partial O_{31}}{\partial {W_{21}}^3}$$
- $$\frac{\partial L}{\partial {W_{11}}^2} = \frac{\partial L}{\partial O_{31}} * \frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial {W_{11}}^2}$$
- $$\frac{\partial L}{\partial {W_{11}}^1} = \frac{\partial L}{\partial O_{31}} * \left(\frac{\partial O_{31}}{\partial O_{21}} * \frac{\partial O_{21}}{\partial O_{11}} + \frac{\partial O_{31}}{\partial O_{22}} * \frac{\partial O_{22}}{\partial O_{11}}\right) * \frac{\partial O_{11}}{\partial {W_{11}}^1}$$

(When computing gradients for backpropagation using chain rule, consider path of flow of the feature)

50.9 TRAINING AN MLP: MEMOIZATION

In Computer Science we have a powerful idea called as Dynamic Programming. This helps us calculate the value of a variable only once. Compute anything only once and store that in a dictionary for reuse.

While computing gradients, gradients such as $\frac{\partial L}{\partial O_{31}}$ can be seen to occur multiple times. Without storing its value we end up re-calculating its value again and again. This will impact the time of computation of all the gradients in an MLP. With memoization we calculate the value of each and every gradient only once. This will avoid repeated calculations while keeping run time minimum.

Chain rule + Memoization → Back Propagation

50.10 BACKPROPAGATION.

Given D = {x_i, y_i}

1. Initialize weights → randomly (there are effective initialization methods available)
    - Input x_i
2. For each x_i in F
    - Pass x_i forward through network
        - Get y_i_hat
        - This is forward propagation
    - Compute loss 2(y_i, y_i_hat)
    - Compute derivatives using chain rule and memoization
    - Update weights from end to start

$$W_{new} = W_{old} - \eta * \left(\frac{\partial L}{\partial W}\right)_{W_{old}}$$

Forward propagation → Using x_i to calculate y_i and L

Backward propagation → Using L to update weights

Both combine to form an epoch

3. Repeat step 2 until convergence, i.e. W_new ~ W_old

1 epoch of training → pass all of the data points through the network once.

Backprop: 'init weights', 'in each epoch (forward prop, compute ;pss, compute derivative(chain rule + memoization), Update weights', 'repeat till convergence'

- Backprop works only if activation function is differentiable (required to update weights)
    - Speed of training depends on computing the derivatives.
    - A faster way is to pass a batch of input data points to speed up training.
        - Batch size depends on RAM capacity
        - Generally we use batch size = 8, 16, 32, 64,... to take advantage of the RAM architecture

---

50.11 ACTIVATION FUNCTIONS

Most popular (Pre-Deep Learning - 1980s and 90s):

- Sigmoid and tanh

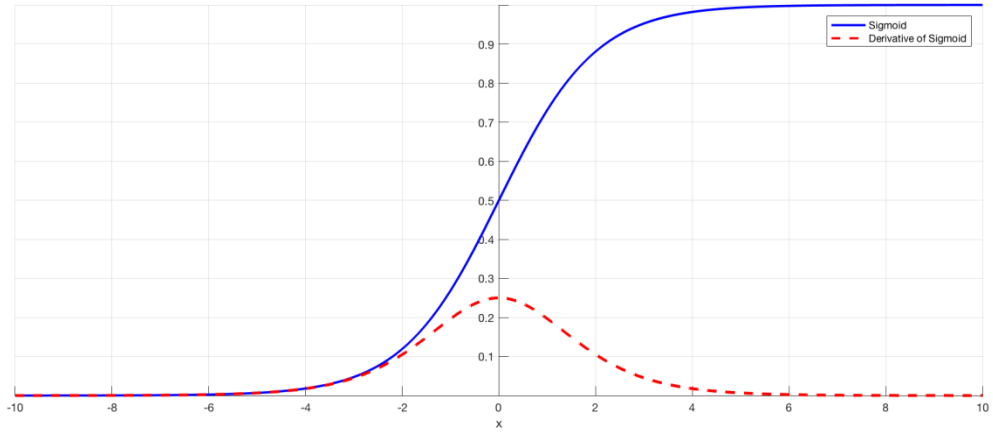
Sigmoid: $\frac{1}{1+e^{-x}}$ [Used in Logistic regression for squashing values]

tanh: $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

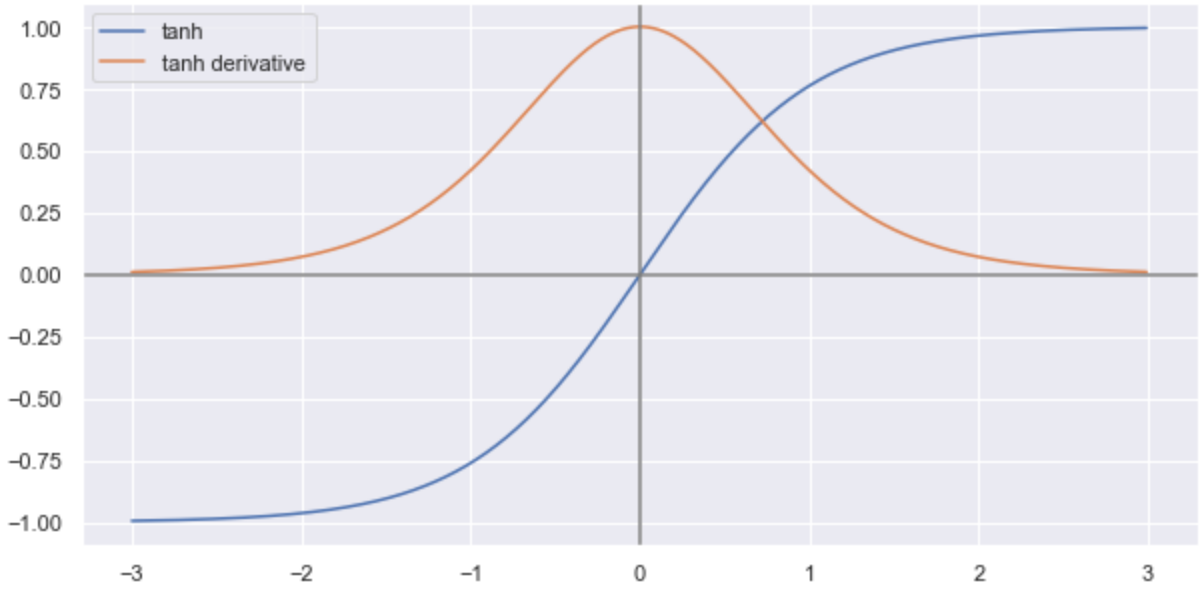
Activation functions should be differentiable and should be faster to compute

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

1. $\sigma'(z) = \sigma(z)*(1-\sigma(z))$



2. Tanh, derivative: 1- tanh sq.



Sigmoid and tanh:

→ differentiable

→ faster to compute gradients

ReLU became most popular activation function as sigmoid and tanh resulted in vanishing gradients

---

50.12 VANISHING GRADIENT PROBLEM

Due to chain rule, multiplication of derivatives which are <1 will result in overall derivative to be small.

→ This will not help weight updates (no training)

→ W_new ~ W_old

- This happened due to sigmoid and tanh activation functions
- People were not able to train deep neural networks due to this vanishing gradients problem.

- Exploding gradients happen when each gradient is >>>1
    - Weight update will be noisy
    - Large update → no convergence

→ training does not stop

Vanishing gradients - no weight updates

Exploding gradient - no convergence

(With sigmoid function, which results in values between 0 and 1, exploding gradients occur when weights are greater than 1)

50.13 BIAS-VARIANCE TRADEOFF.

---

1. As number of layers increase, we will have more weights, leading to overfit → low bias, high variance
2. Logistic Regression model - fewer weights (compared to MLP) - leads to underfitting - high bias

- MLPs typically overfit on train dataset
- We avoid this using Regularizers (L1 or L2 or Dropouts)
    - We will have 'lambda' coefficient of regularization term as a hyperparameter. Also number of layers is a hyperparameter
- Using regularization, we reduce variance

50.14 DECISION SURFACES: PLAYGROUND

Site: playground.tensorflow.org - A web browser interface to understand variations due to change in hyperparameters.

51.1 DEEP MULTI-LAYER PERCEPTRONS:1980s TO 2010s

Until 2010, people were trying to build 2 to 3 layered networks due to: vanishing gradients, limited data (easy to overfit, no weight updates -> no training), limited compute power.

By 2010 we got lots of data (Internet, labelled data (quality)), compute power has increased (Gaming GPUs - NVIDIA - found to be suitable for Deep Learning), advancements in algorithms. This paved way for modern Deep Learning achievements.

With classical ML (Mathematician approach), theory was first built and then proved through experiments. With DL it became possible to experiment (cheap) with different ideas first (Engineer Approach) and then develop theory.

> Why not use Deep Learning for every problem?

Sometimes DL works better sometimes ML works better, it is generally experimented to see whose performance is good.

Depending on data, if the data is simple (such as having an underlying Linear Relationship) using ML and DL will be same and adding DL which is compute expensive does not give advantage over ML, while with complex data DL may work better.

But EDA and other experiments are required to be performed to decide which Algorithm will perform well.

> So in short, mlp's are very complex mathematical functions with different variables(features) and weights are their coefficients such that we will find optimum values for weights which can accurately predict the result. Since weights are adjusted to the patterns in the train data and could predict the output if a similar pattern(data point) has been injected into it.

Small dataset leads to overfit when:

- Increasing the number of hidden units and layers
- Increasing number of epochs

> Neural Networks handle outliers by using batch training and robust cost functions that include regularization.

51.2 DROPOUT LAYERS & REGULARIZATION.

Deep NN -> generally occurring problem in overfitting; Regularization such as L1 and L2 can be used, dropout is prefered regularization equivalent technique.

Concept of dropout can be linked to the sampling of subset of columns in RandomForest. Each tree looks at a random subset of features. **Regularizing through a Random subset of features.** Base learners are high variance and low bias models. Through regularization we are reducing variance.

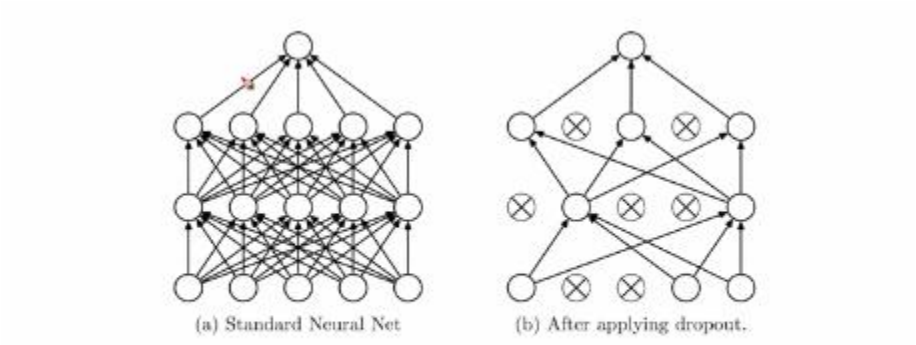


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Random Neuron Dropout per epoch with a dropout rate (percentage of neurons dropped)

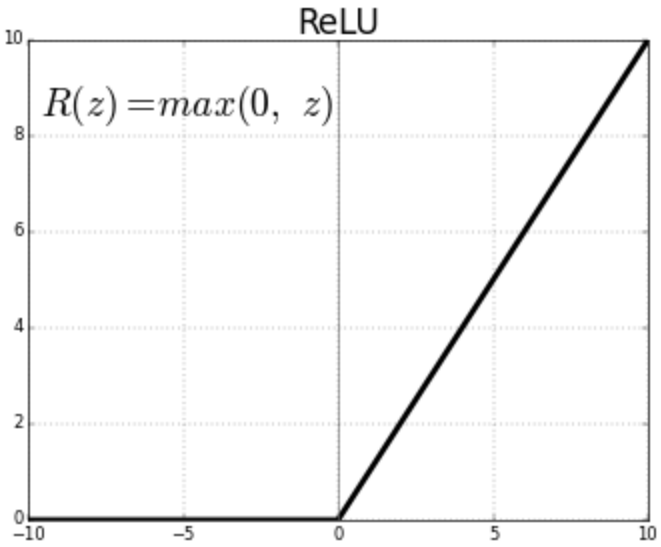
At training time neuron is present with probability ‘p’, at test time all neurons are present but to account of probability of presence, multiply weights of dropout with p;

High dropout rate -> low keep probability -> underfitting -> Large regularization

Low dropout rate -: high keep probability -> overfitting -> small regularization

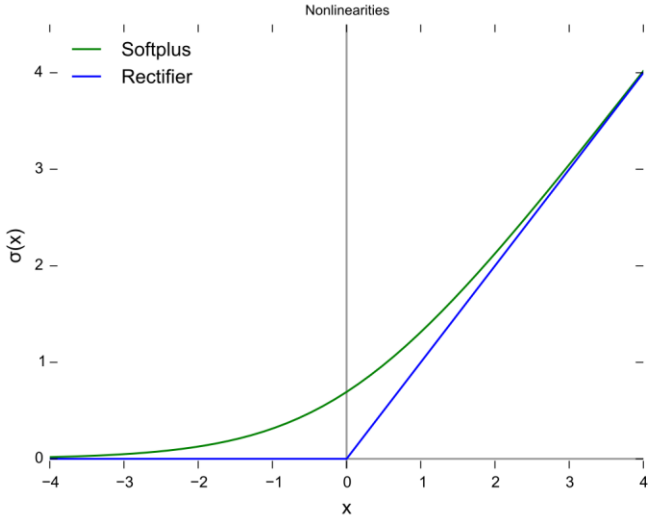
51.3 RECTIFIED LINEAR UNITS (RELU).

- Classical NNs: Vanishing gradients: small updates
- ReLU: becam by default activation function:



- f(Z) = max(0,Z)
- f'(Z) = {0 if Z <0 or 1 if Z>0) (No vanishing gradient or No exploding gradients but there is dead activation below 0 (need to monitor dead activations))
- Relu function is non-linear (as function changes at 0), not differentiable at 0.
- As derivative computing is easy and computationally
- ReLU converges faster than tanh (due to absence of vanishing gradients)

- As ReLU is not differentiable at 0, a smooth approximation for ReLU is defined to take care of non-differentiability problem
- Softplus: f(x) = log(1 + exp x); f'(x) = sigmoid(x)



- ReLU is simpler and computationally efficient. Derivatives can be computed with if else block.

- Leaky ReLUs: for z<=0 f(z) = 0.01z, f'(z) = {1 for z>0, 0.01 else} (can result in vanishing gradients) used when a lot of relu activated neurons become dead.

- ReLU Advantages:
  - Faster convergence
  - Easy to compute
- ReLU limitations:
  - Non-differentiable at zero
  - Unbounded
  - Dying relu
  - Non-zero centered

a. Why non-linear activations?

   If the activation functions are linear, a deep neural network will be similar to a single layer neural network which cannot solve complex problems. While non-linearity gives a good approximation of underlying relations.

b. When do we use sigmoid function?

   Generally, at the last output layer in case of binary classification problems, Softmax in output layer is used for multi-class classification problem, linear for regression.

51.4 WEIGHT INITIALIZATION.

- Logistic Regression: initialization weights randomly, from a random normal distribution
- If init all weights to zero or to same value: if activation functions across all neurons in a layer are same, all neurons compute the same thing, same gradient updates no training -> problem of symmetry
- If weights distribution is asymmetric, all the neurons will learn different aspects of the input data
- If init all weights to large -ve numbers: ReLU(z) will be equal to 0 (normalizing data is mandatory: mean centering, variance scaling): we will have vanishing gradients
- For sigmoid activation at extremities we have vanishing gradients

a: Normal distribution

- Weights should be small (not too small)
- Not all zero
- Good-variance (each neuron will learn differently)

- Weights are random normally initialized

Can we have better init strategies?

(fan_in for a neuron = number of input connections, fan_out = number of output connections: Weights are suggested to be initialized by fan_in and fan_outs, no concrete agreement which works well)

b: Uniform initialization:

- Weights are initialized to a Uniform distribution
  - Unif[-1/sqrt(fan_in),1/sqrt(fan_in)], selection of each value is equally likely

c: Xavier/Glorot init (2010) - useful for sigmoid activations

- Normal: Initialize to a mean centered normal distribution with variance (sigma sq.) = 2/(fan_in + fan_out)
- Uniform: Initialize to
  - ~Unif[(-sqrt(6)/sqrt(fan_in+fan_out), sqrt(6)/sqrt(fan_in+fan_out)]
- Using fan_in and fan_out also

d. He init - ReLU activations

- Normal: Initialize to a mean centered normal distribution with variance (= sigma sq.) = 2/(fan_in)
- Uniform: ~Unif[(-sqrt(6)/sqrt(fan_in), sqrt(6)/sqrt(fan_in)]

51.5 BATCH NORMALIZATION.

- Let us take: an MLP fully connected,
- Generally, we perform pre-processing of data by normalization (mean centering and variance scaling), even then for a deep MLP, we input a mini batch of data repeatedly, a small change in input leads to a large change in the last layer of a Deep Neural Network (though everything looks good near input layer)
- Thus normalization is used to reduce changes deep in the network
- This problem is called Internal co-variance shift: “change in the distribution of network activations due to the change in network parameters during training”
- BN: normalization with batch mean and batch variance and then **scaling and shifting**. Parameters gamma, beta: gamma*(x_norm) + beta; these are learnt by backpropagation.
- BN: faster convergence, allows using large learning rates, weak regularizer and avoids internal covariance shift

BN on test set: for training: batch mean and variance, for test: train population mean and variance

Additional material: https://arxiv.org/pdf/1502.03167v3.pdf

51.6 OPTIMIZERS: HILL-DESCENT ANALOGY IN 2D

- Objective was to minimise loss functions (Optimization approach)
- In neural networks, Gradient Descent cannot solve all problems very well

Working principle of optimizers using: Hill -descent analogy

- Task: min L(w)

  Case 1: W = Scalar

  - If L(w) is y = x**2/4a:

    There is no maxima and has only one minima at 0

  - If L(w) is a complex:

    We can have local minima, local maxima, global minima and global maxima, at minima or maxima we have zero gradient

    Update function: W_new = W_old - eta (grad(L,W))

    Saddle point: gradient is also zero, but not a minima nor a maxima

    Zero gradient implies minima, maxima or saddle point

    SGD could get stuck at saddle point

  - Convex and Non-convex functions: Convex have one minima or maxima

    Non-convex have multiple local minima or local maxima

Logistic Regression, Linear Regression and SVM functions: have loss functions that are convex, for DL-MLPs: loss functions are non-convex thus initialization of weights determine the minima or maxima we reach, depending on the initialization we can reach a global minima or local minima

Additional Material:

- Squared loss is used as its derivative is a linear function which will have a unique solution, while derivatives of higher order loss functions will lead to having more than one solution or multiple minima or maxima locations, Squared loss can generally be preferred as it will only have one minima or maxima

Why does DL have non-convex loss functions?

Loss function of DL models: dice loss, cross-entropies, MSE. These are defined by the programmer, mostly non-convex functions are used as their performances are better and the choice depends on architecture and the network output or our desired target values. Loss functions represent the Network's output function.

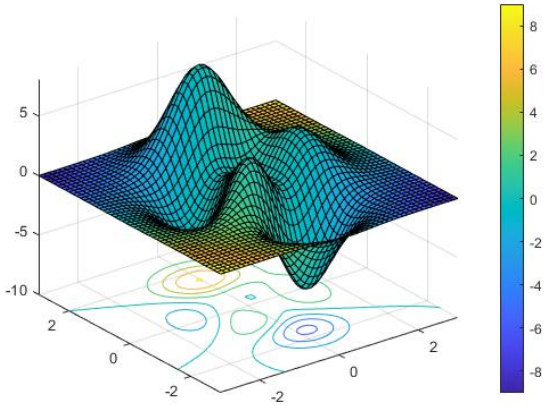
51.7 OPTIMIZERS: HILL DESCENT IN 3D AND CONTOURS.

Case 2: Let W be 2 dimensional (like a catching net)

Let loss function be convex

Contour: project all points of a 3D plot at same height to a plane, visually indicates location of maxima and minima

(Graphically shown)



Simple SGD will get stuck at saddle point, local maxima and local minima

---

51.8 SGD RECAP

Update: W_k_ij_new = W_k_ij_old - eta*(grad(L,W_k_ij_old))

Generalized form: W_t = W_t-1 - eta*grad(L, W_t_1)

grad(): computed using all points: Gradient Descent

computed using one point (random): Stochastic GD

computed using k points (random sample): mini-batch SGD (preferred)

grad(): mini-batch SGD ~ GD (mini batch SGD approximation to GD, and is compute efficient) (mini-batch SGD is a noisy descent)

Task: denoise gradients for SGD

51.9 BATCH SGD WITH MOMENTUM.

SGD updates are noisy as compared to GD updates,

Task: denoise SGD updates:

With SGD we have gradients at each time step as: a1, a2, a3, …

- a_i is an original gradient calculated using sgd, v_i is exponential average of these gradients

a. Applying average method of denoising (r = gamma):

V1 =a1, V2 = gamma*V1+a2, V3 = gamma*V2+a3, ...

a1, r*a1+ a2, r*(r*a1 + a2) + a3, …..: a.k.a exponential weighted average

W_t = W_t-1 - eta*grad (mini batch SGD)

Exp. weighting: V_t = r* V_t-1 + eta * g_t

Recent gradients has more influence

If gamma = 0, mini batch SGD

If gamma = 0.9, W_t = W_t-1 - eta*V_t-1 - eta*(g_t).

Momentum adds to convergence by travelling more distance towards optimum by getting to know the sense of direction of movement

51.10 NESTEROV ACCELERATED GRADIENT (NAG)

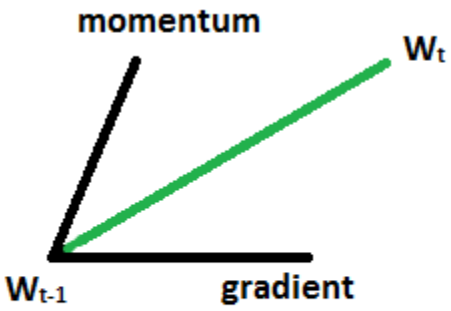
Previous Lecture: SGD + momentum: exponential gain to create momentum

Gradient: eta * g_t, Momentum: gamma*V_t-1

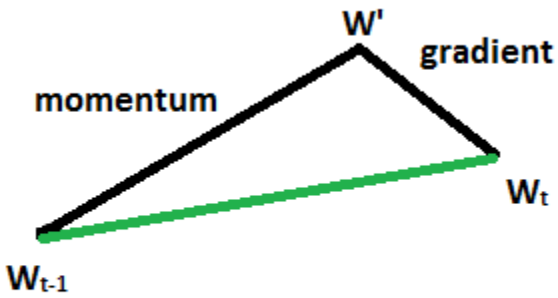
SGD: g = grad(L,W) [W_t-1]

NAG: First move in momentum direction, then compute gradient and then move in gradient direction: W_t = W_t-1 - gamma * V_t-1 - eta*g'

NAG: g' = grad(L,W) [W_t-1 - gamma * V_t-1]



SGD + momentum



Nesterov Accel. Grad

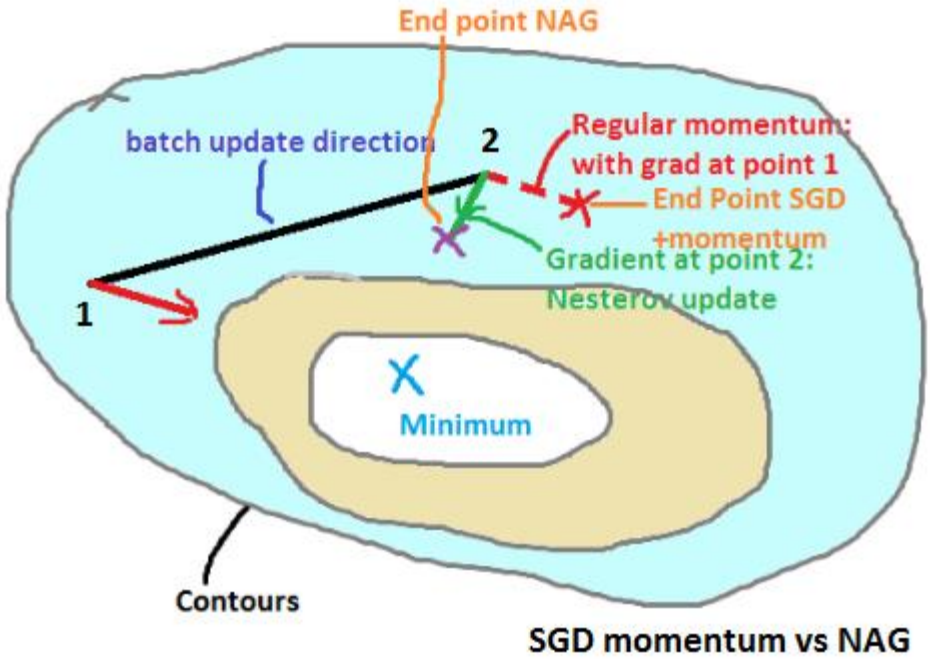


Figure made using Paint

NAG update ends up slightly closer to the optimum, convergence becomes faster compared to SGD + momentum.

51.11 OPTIMIZERS:ADAGRAD

- Adagrad: Introducing adaptive learning rate for each weight
- Some features in data points can be sparse.
- SGD: W_t = W_t-1 - eta*g_t: eta same for all weights
- Adagrad: W_t = W_t-1 - eta'_t*g_t: (eta'_t = different eta for each weight for each iter)
- eta'_t = eta_t/(sqrt(alpha_t + epsil.)
- alpha_t = sum[i = 1 to t-1](g_i**2) : g_i = grad(L,W)[W_t-1]
- aplha_t >= alpha_t-1 → eta'_t <= eta'_t-1

- As ‘t’ increases, the learning rate (for each weight) decreases. Learning rates are different for each weight and this learning rate decay happens adaptively in the case of Adagrad

Advantage:

+ Adaptive learning rate: no manual decay needed
+ Right learning rates for different features

Disadvantage:

- alpha increases as t increases, becomes a large value which makes learning rate small, resulting in negligible updates → no training

Learning AdaGrad makes it easy to understand other adaptive optimizers

Why different learning rates for sparse features?

Link: https://youtu.be/c86mqhdmfL0

Sparse features makes gradient small (due to summation on all data points)

Update does not happen, thus eta or the Learning rate requires to be larger for these sparse features, while for dense feature gradient will not be small. Simply put, constant learning rates will only help the weights to update with contributions from dense features and not from sparse features. Weights updates for sparse features will be negligible as gradients will be small which demands for a higher learning rate. If a constant higher learning rate is used, the weights are updated with large steps and convergence will never happen. Thus we require different learning rates. **Sparse features demand large learning rates, while dense features require smaller learning rates.**

51.12 OPTIMIZERS : ADADELTA AND RMSPROP

Previous Lecture: Adagrad’s alpha can become very large resulting in slow convergence.

- eta’_t = eta_t/(sqrt(alpha_t + epsil.)

  alpha_t = sum[i= 1 to t] (g_i **2)

Adadelta: W_t = W_t-1 - eta’_t * g_t

eta’_t = eta / (sqrt(eda_t + eps.))

eda(0) = 0

eda_t = gamma * eda_t-1 + (1 - gamma) * (g_t-1) **2

gamma = 0.95, generally

Adadelta: take exponential weighted averages of gradient of squares instead of simple sum to avoid large alphas avoiding slow convergence.

51.13 ADAM

Adaptive Moment Estimation: most popular DL optimizer,

In Adadelta we are using eda(g_i squares)

Storing eda of (g_i);

Analogy in Statistics: Mean: first order of moment

Variance: second order of moment

m_t = beta_1 * m_t-1 + (1 - beta_1) *g_t

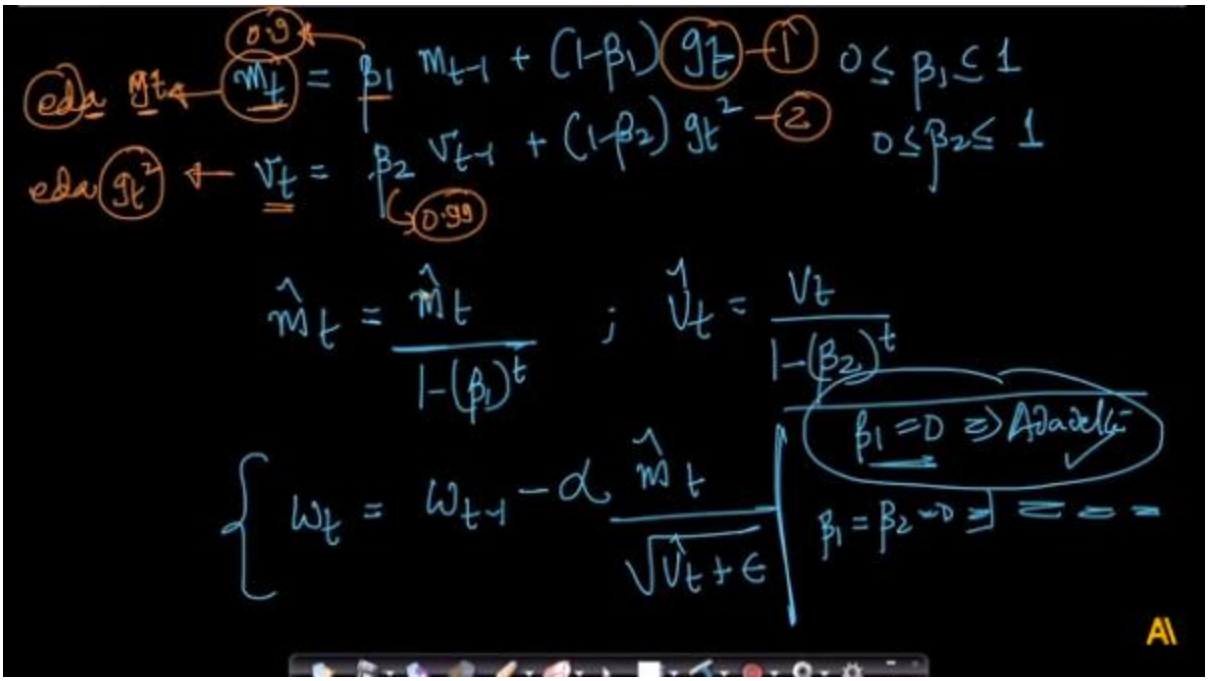
v_t = beta_2 * v_t-1 + (1 - beta_2) * (g_t **2)

m_t_hat = m_t / (1 - beta_1**t)

v_t_hat = v_t / (1 - beta_2**t)

w_t = w_t-1 - alpha * (m_t_hat/ sqrt(v_t_hat + eps.))

t = time step




```
for t in range(num_iterations):
    g = compute_gradient(x, y)
    m = beta_1 * m + (1 - beta_1) * g
    v = beta_2 * v + (1 - beta_2) * np.power(g, 2)
    m_hat = m / (1 - np.power(beta_1, t))
    v_hat = v / (1 - np.power(beta_2, t))
    w = w - step_size * m_hat / (np.sqrt(v_hat) + epsilon)
```

51.14 WHICH ALGORITHM TO CHOOSE WHEN?

Type of Optimizers:

Mini batch SGD, NAG, Adagrad, Adadelta, RMSProp, Adam we came across

Source: Link

Considering getting stuck at saddle points or at local minima or local maxima (loss does not change):

- Mini Batch SGD works well for Shallow NN.
- NAG works well for most cases but slower than Adagrad, Adadelta and Adam
- Adagrad - sparse data

- Adam works well in practice - favored optimizer - fastest algorithm for convergence
- For sparse data adaptive learning rates are required.

51.15 GRADIENT CHECKING AND CLIPPING

When we train MLPs, we need to monitor Weight updates. Thus we need to check gradients (each epoch and each weight). This will help us detect vanishing gradients which easily occurs in the first few layers due to farness from the output layer (gradients become too small or very large as we move far from the output layer towards input layer).

For exploding gradients: we can use gradient clipping: All the weights or gradients are stored in a single vector.

For example we can have an L2 norm clipping:

All the gradients in the gradient vector are divided by the vector's L2 norm (sum of squares of all gradients). This will clip the gradients to 1. And multiplying these with a threshold value we clip all the gradients to the threshold value.

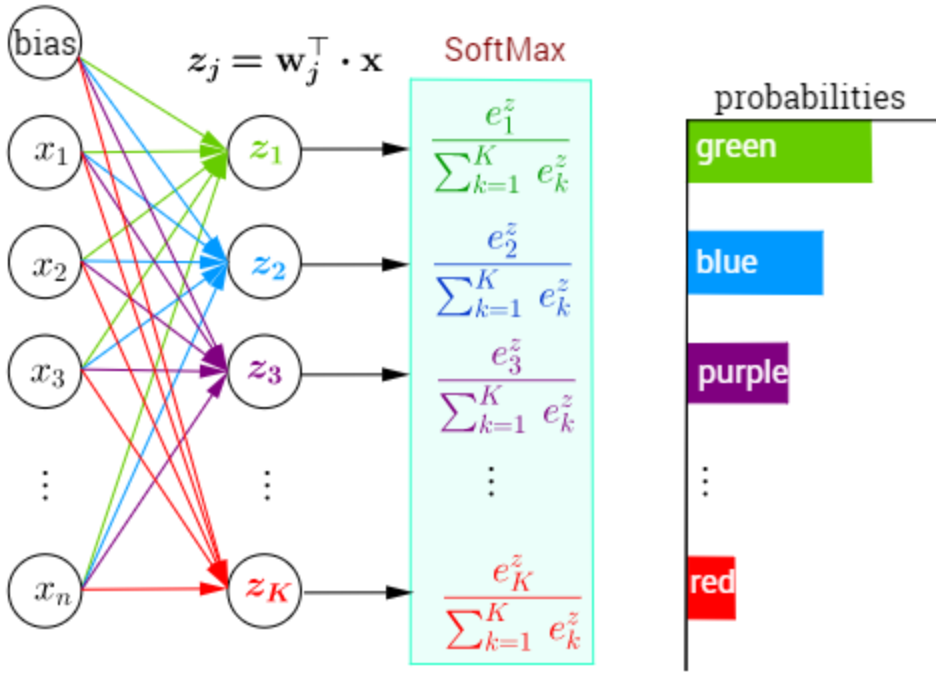
Rule of thumb: Monitor Gradients to ensure training is proper

51.16 SOFTMAX AND CROSS-ENTROPY FOR MULTI-CLASS CLASSIFICATION.

Logistic Regression: Sigmoid activation: binary classification

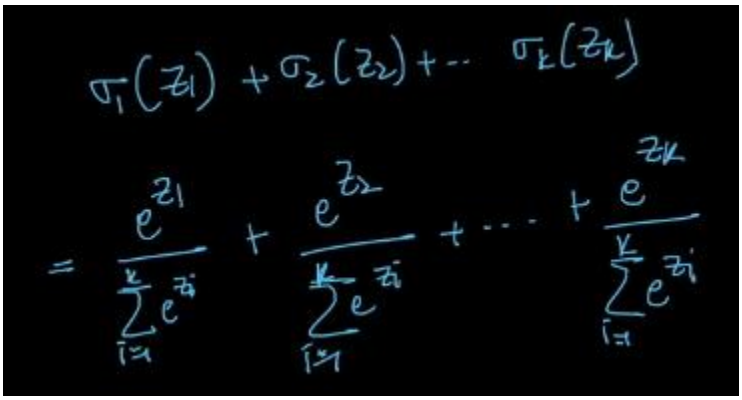
For multi-class classification using Logistic Regression we use One versus Rest method. But can we do something else, like extending the basic math of Log. Reg. Extension of Logistic regression to Multi-Class classification results in softmax. Recap: Output of the Logistic regression network is the probability of y_i = 1 given x_i.

P(y_i = 1|x_i) = 1/(1+ e**(-W.T *x).



Summation of probabilities = 1 in Softmax activation

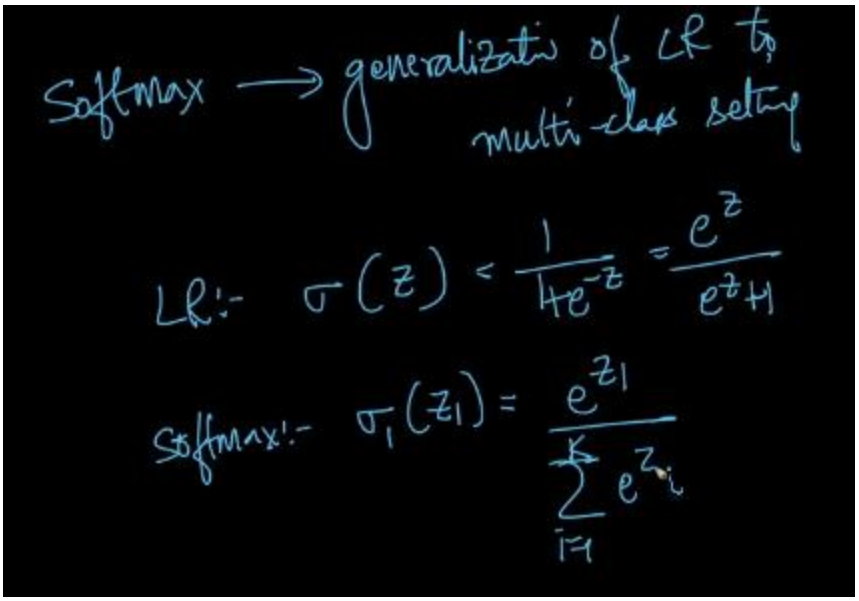
Softmax is a generalization of logistic regression for multi class problems.



sigma_1(z_1) = P(y_i=1|x_i); sigma_2(z_2) = P(y_i=2|x_i);

sigma_3(z_3) = P(y_i=3|x_i); sigma_4(z_4) = P(y_i=4|x_i);

Summation = 1



In Logistic Regression, we optimize log loss for binary classification,

where L_i = y_i*log(p_i) + (1-y_i)*log(1-p_i)

In Softmax, we optimize multi-class log loss: $-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log p_{ij}$

y_ij is 0 for all classes other than true class which has y_ij = 1

Regression: Squared Loss, 2 class classification: 2 class log loss, k-class classification: Multi class log loss

---

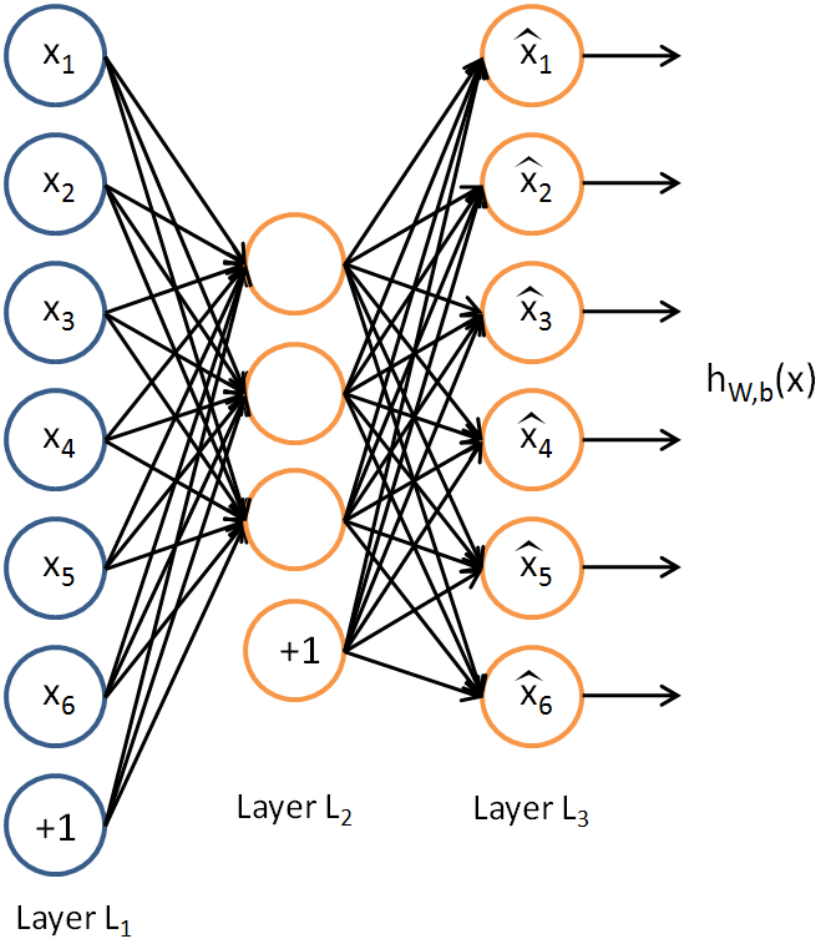
51.17 HOW TO TRAIN A DEEP MLP?

- **Preprocess data: Data Normalization**
- **Weight Initialization:**
    - **Xavier/Glorot for sigmoid or tanh**
    - **He init for ReLU**
    - **Gaussian with reasonable variance**
- **Choose right Activation functions: ReLU (and its variations: avoids vanishing gradients)**
- **Add Batch Normalization layers for Deep MLPs for later layers: helps you tackle internal co-variance shifts, Use Dropouts for regularization in deep MLPs**
- **Optimizer choice: Adam general choice**

- **Hyperparameters: Architecture:Number of layers, number of neurons, dropout rate, optimizer hyperparameters**
- **Loss function: Log loss or Multi-class Log loss or Square loss (easy differentiable)**
- **Monitor gradients (Apply gradient clipping if needed)**
- **Plot Loss vs epoch and other metric plots**
- **Remember to avoid Overfitting (Early Stopping, Dropouts, etc.)**

51.18 AUTO ENCODERS.

Neural network which performs dimensionality reduction (better than PCA and tSNE sometimes) (tSNE tries to preserve the neighborhood)



Dataset: D = {x_i} x_i E R(d) : an unsupervised problem

To convert D to D’ such that X_i E R(d’ << d)

Given x_i: as we reduce the number of neurons in the next layer dimensionality reduces.

Let us take a simple encoder with three layers: Input Layer: 6 neurons, Hidden: 3 neurons, Output: 6 neurons

> x_i with 6 dimensions is given as input, it is compressed to 3 dimensionality in the hidden layer, then at output layer it increases the dimensionality to original input. If the output is similar to the input then the representation of the input at the hidden layer is good. This implies that the reduction in dimensionality has preserved information of the input.

Thus if we have input ~ output in an auto encoder, then we have the compression part reducing the dimensionality without losing variance or information.

Denoising autoencoder: Even though we have noise in input, at the output we will get denoised representation of the input as the dimensionality is reduced in the intermediate hidden layers.

Sparse autoencoder: Getting sparse output by using L1 regularizer

If linear activations are used or a single sigmoid hidden layer, the optimal solution to an autoencoder is closely related to PCA.

51.19 WORD2VEC : CBOW

Ways of Featurizing text data: BOW, TFIDF, W2V

Ex: The cat sat on the wall.

> Focus word: sat: The, cat, on, the, wall are context words
>
> If Focus word: cat: The, sat, on, the, wall are context words

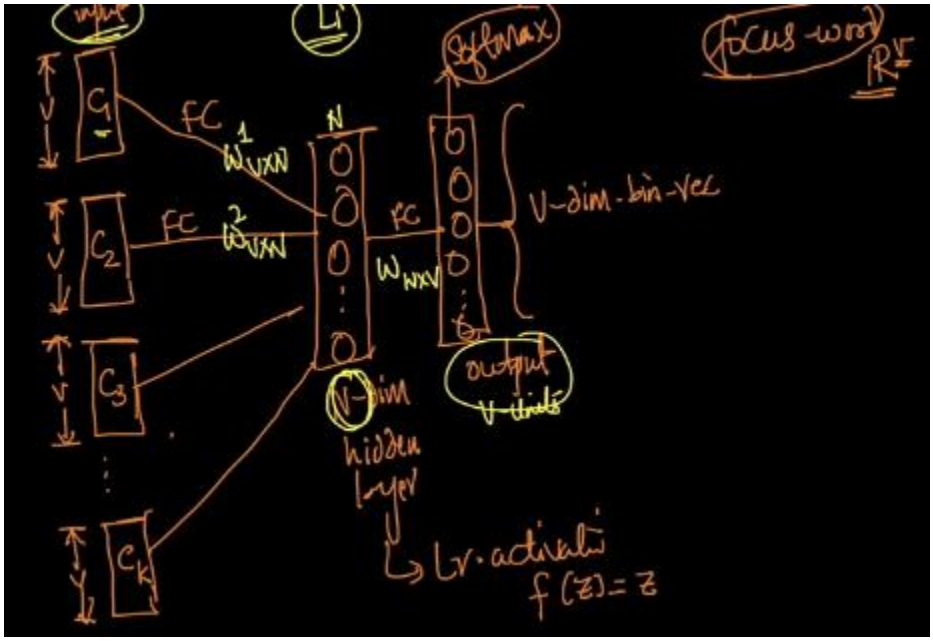
Idea: context words are useful for understanding focus words and vice-versa.

There are two algorithms: CBOW, SkipGram

CBOW: Continuous Bag of Words

We have a dictionary/vocab; Use one hot encoding for each word based on vocab length. Core idea behind CBOW: Given context words can we predict focus words. Linear or Identity activation is used. Softmax is used at the output layer.

Structure:



Take all of the text dataset, create focus word - context words dataset, train the neural network with above structure on this dataset.

51.20 WORD2VEC: SKIP-GRAM

CBOW: Predict focus word given context words

Skip-gram: Predict context words given focus word

Input: v-dimensional one hot encoded focus word. Hidden layer with N dimension with linear activations. Use multi output Softmax layers which gives context words as output corresponding to each output layer that are stacked to form a single output layer (k softmax layers).

# of weights: CBOW: (K+1) (NxV), 1 softmax: performs well for frequently occuring words

Skipgrams: (K+1) (NxV), k softmax: slow training: performs well for less occuring words.

51.21 WORD2VEC :ALGORITHMIC OPTIMIZATIONS.

CBOW and SkipGrams: Millions of weights: time taking

Hierarchical Softmax: Using binary tree to predict a word.

Negative sampling: Update a sample of words: Update weights to all the target words and weights of some of non target words (selection of non target words is based on probability value corresponding to the occurrence of the word).

52

DEEP LEARNING: TENSORFLOW AND KERAS.

52.1 TENSORFLOW AND KERAS OVERVIEW

Tensorflow/ Keras: Tool kits or libraries that enable us to code for Deep Learning

Tensorflow: most popular DL library, open sourced by Google, by Nov, 2015

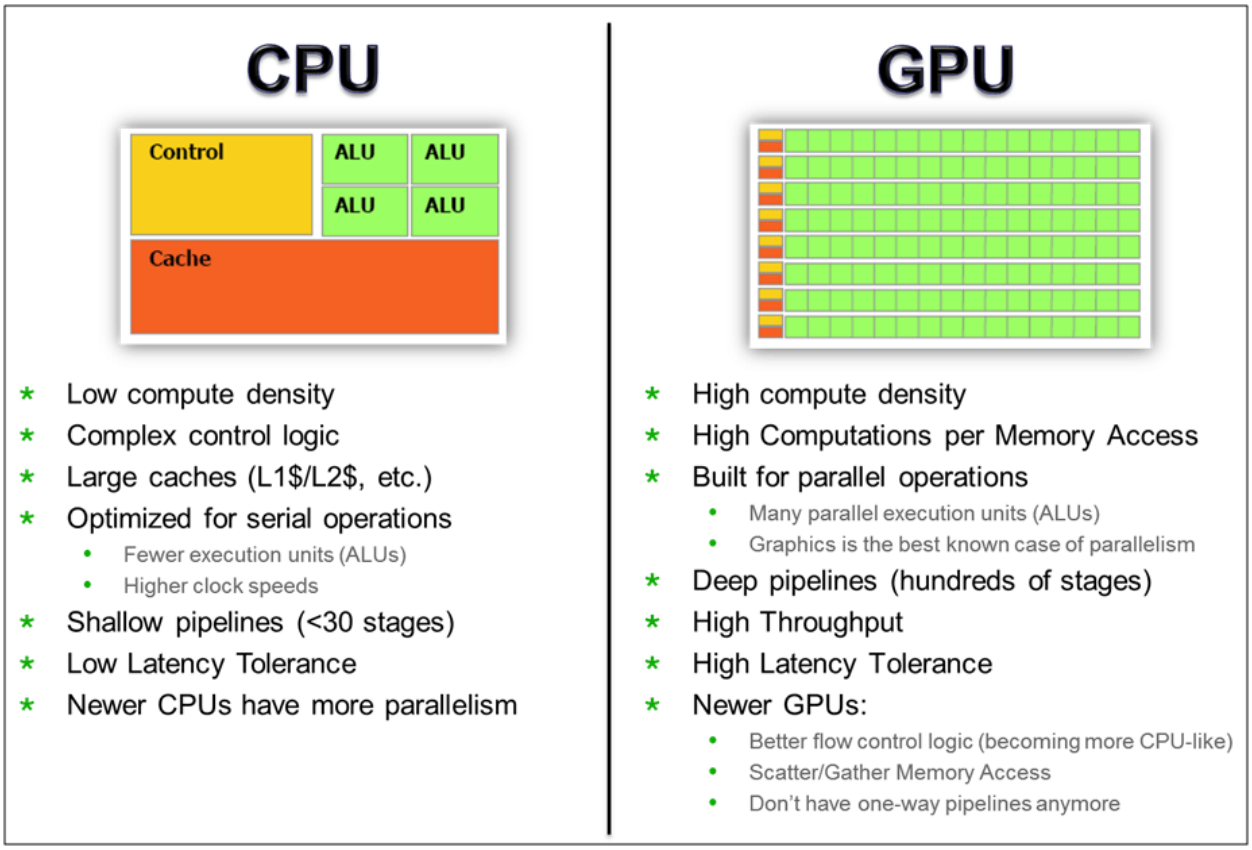
- Helps researchers, for developers and for deployment engineers also (two different tasks)
- Core of Tensorflow was written in C/C++ for speed, they made interfaces available in Java, Python and JavaScript
- Tensorflow Lite can run on Android for deployment on Phone.
- Tensor: Mathematical term for vectors, 1D vector : 1D tensor, 2D vector : 2D tensor, 3D vector : 3D tensor
- Deep Learning is all about tensor operations
- Flow may be inspired from forward and backward propagation (flow of data)
- It gives lot of low-level control of models
- With Keras: simple (similar to SKLearn); high level NN library: faster for developing and deploying models; less lines of code
- Keras is front end: Tensorflow is backend; other backend: Theano, Pytorch, Caffe, MXNET
- Tensorflow: Research easy, Keras: Development easy

52.2 GPU VS CPU FOR DEEP LEARNING.

Graphics Processing Unit: Became suitable for Deep Learning

CPU: Central Processing Unit

GPUs: 1990s; for gaming (Graphic cards);



RAM is connected to CPU through motherboard, Cache is like RAM on chip, The processors can communicate with Cache faster than with RAM, With multiple cores multiple calculations are done at time parallely. Around 2GHz speed

GPU: Multiple processors: 1024, 512, etc.Each GPU core is slower than a CPU core, around 400MHz speed. Every unit has a processor and a cache.This is a distributed structure. Sum of all cache in GPU is called as VRAM (Video RAM). Takes data from RAM, distributes across all units, and the processor unit works very fast on its corresponding Cache data.

GPUs are fast if we have parallelizable tasks such as doing Matrix operations as each of the calculations are not dependent among themselves. Because of these characteristics of GPUs, Deep Learning is experiencing developments.

52.3 GOOGLE COLABORATORY.

Provided by Google for Machine Learning education and research;

Link: colab.research.google.com

Similar to Jupyter Notebook

Cloud computing: shared computing resources; bunch of computers tied together for ready access over internet

Google Colab computers are extremely powerful

52.4 INSTALL TENSORFLOW

https://www.tensorflow.org/install/install_windows

52.5 ONLINE DOCUMENTATION AND TUTORIALS

- https://www.tensorflow.org/get_started/
- https://learningtensorflow.com/
- https://cloud.google.com/blog/products/gcp/learn-tensorflow-and-deep-learning-without-a-phd

52.6 SOFTMAX CLASSIFIER ON MNIST DATASET

Using Tensorflow:

MNIST dataset: Given image predict its class

a. Get data from Tensorflow datasets

b. Import Libraries, check GPU, CPU capacities

c. Placeholder: memory location

Placeholders and Variable:

x = tf.placeholder(tf.float32, [None, 784])

Constant: cannot be changed

Variable: Updated during training or other computations

A placeholder can be imagined to be a memory unit; where values of some data points are stored for computations.

W and b are variables

- W = tf.variable(tf.zeros([784,10]))
- b = tf.variable(tf.zeros([10]))
- y = tf.nn.softmax(tf.matmul(x,W)+b)
- y_ = tf.placeholder(tf.float32, [None.10])
- cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_*tf.log(y), ...)
- reduce_sum: takes tensor of 2D reduces to 1D and applies sum
- then training, etc.
- sess = tf.InteractiveSession() : initiates a computational graph
- tf.global_variables_initializer().run()
- Run loops over batches of training dataset
- Metrics evaluations
- Termination criteria: loss directed termination

52.7 MLP: INITIALIZATION

For MNIST dataset

a. Importing libraries

b. Dynamic plot: at every epoch

c. Architecture used: 784 (FC) – 512 (FC) – 128 (Softmax) – 10

d. Placeholders, Variables, Weight initializations,

52.8 MODEL 1: SIGMOID ACTIVATION

52.9 MODEL 2: RELU ACTIVATION.

52.10 MODEL 3: BATCH NORMALIZATION.

52.11 MODEL 4 : DROPOUT.

52.12 MNIST CLASSIFICATION IN KERAS

https://drive.google.com/file/d/1tIEOPJiJtMzStFai47UyODdQhyK9EQnQ/view

52.13 HYPERPARAMETER TUNING IN KERAS.

1. Definition of keywords

Hyperparameter: A parameter of learning algorithm that is not of the model, these remain constant during model training. The values of other parameters are derived from training. Example of hyperparameter: number of neurons in each layer. Example of model parameter: learning rate. Given hyperparameters the training algorithm learns the parameters from training.

https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)

Hyperparameter tuning: Through tuning we derive a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given test data. It

is the procedure through which we derive hyperparameters that yields an optimal model. This model is further trained on training set to update weights.

2. Detailed explanation

Neural Networks are so flexible that they a drawback in terms of hyperparameters which can be tweaked. A simple MLP can have lots of hyperparameters: Number of layers, number of neurons per layer, type of activation function to use in each layer, weights initialization logic, and so on. One option is to try different combinations of hyperparameters and see which combination works best on validation set. Use SKLearn's GridSearchCV or RandomizedSearchCV. Wrap keras models to mimic regular SKLearn classifiers. SKLearn tunes hyperparameters for maximum score by default. Thus loss functions needs to be transformed into a metric. This way of tuning is generally time consuming. Efficient toolboxes such as hyperopt, hyperas, skopt, keras-tuner, sklearn-deep, hyperband and spearmint are available for this purpose. As the search space is huge, follow the following guidelines to restrict search space:

For number of hidden layers: Slowly add number of hidden layers from 1 to 50 or 100 until overfitting on training set. Above this use, transfer learning.

Number of neurons per layer: For input and output layers it is predetermined by data. Randomly pick a number which is power of 2. Use early stopping and regularization to prevent overfitting. It is generally preferred to increase number of layers over number of neurons per layer. Avoid having too few layers to have enough representational power of the data.

Learning rate of loss function: Change learning rate from 10^{-5} to 10^{1} at an interval of epochs. Plot loss function and select the LR that is just behind the value which shows minimum loss value. If minimum is found at 1 use 0.1 as LR.

Optimizers, batch_size and activation functions have a fixed set of choices. While with number of epochs use a large number and use early stopping to stop training.

Especially if after choosing LR, you tweak a hyperparameter, LR should again be tuned. A best approach is to update Learning Rate after tweaking all other hyperparameters.

3. Video Lecture

Multi Layered Perceptrons have a lot of hyperparameters.

a. Number of Layers
b. Number of activation units in each layer
c. Type of activation: relu, softmax
d. Dropout rate, etc.

How do you do hyperparameter tuning for MLP?

- Scikit-learn has two algorithms for this purpose: GridSearchCV and RandomSearchCV, we used them a lot in Machine Learning assignments
- Keras models require to be connected with the above algorithms. Build DL models in keras and use SKLearn's hyperparameter tuning algorithms

Code:	**Procedure:**
# Hyper-parameter tuning on type of activation of Keras models using Sklearn from keras.optimizers import Adam,RMSprop,SGD def best_hyperparameters(activ): ‘’’ Defining model: This function returns a model with activation type = input string Input: string Output: model ‘’’ model = Sequential() model.add(Dense(512, activation=activ, input_shape=(input_dim,), kernel_initializer=RandomNormal(mean=0.0, stddev=0.062, seed=None))) model.add(Dense(128, activation=activ, kernel_initializer=RandomNormal(mean=0.0, stddev=0.125, seed=None))) model.add(Dense(output_dim, activation='softmax')) model.compile(loss='categorical_crossentropy', metrics=['accuracy'], optimizer='adam') return model	- We are tuning only activation type (comparing Relu and Softmax) - Assume: all important libraries are downloaded, datasets loaded and everything is ready for hyperparameter tuning - Model is defined using a function best_hyperparameters(). This is a sequential model. We are using MNIST dataset which has 784 columns. This model has 4 layers. Input layer has 784 neurons. There are two hidden layers with 512 and 128 neurons respectively. The last layer is the output layer with number of neurons = output_dim. Type of Activation is the input to this function. - Model is compiled with categorical_cross_entropy with metrics = accuracy and optimizer = ‘adam’ - The actication choices are softmax and relu. Softmax is just a multi-valued version of sigmoid.

# https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/ 	
activ = ['softmax','relu']
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import GridSearchCV
model = KerasClassifier(build_fn=best_hyperparameters, epochs=nb_epoch,
batch_size=batch_size, verbose=0)
param_grid = dict(activ=activ)
	- Activation type choices: ‘softmax’ and ‘relu’. Wrapper makes connection between keras model and Scikit-learn tuner. We are using KerasClassifier function for this purpose. This makes a wrapping over Keras model and makes it readable by Scikit-learn.
	- KerasClassifier takes model defined which gets compiled. Its arguments include epochs and batch_size.
	- A parameter grid is created in dictionary format.

# if you are using CPU 	
# grid = GridSearchCV(estimator=model, param_grid=param_grid, 	
n_jobs=-1)
# if you are using GPU dont use the n_jobs parameter 	
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid_result = grid.fit(X_train, Y_train)
	- GridSearchCV is defined with KerasClassifier as estimator with a parameter grid.
	- Fit function is called to evaluate the model on each choice of the hyperparameter. Here, activation is hyperparameter for hidden layers and its choices are softmax and relu. The GridSearchCV hyperparameter tuning goes through each of the parameter in the parameter grid.

print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_)) means = grid_result.cv_results_['mean_test_score'] stds = grid_result.cv_results_['std_test_score'] params = grid_result.cv_results_['params'] for mean, stdev, param in zip(means, stds, params): print("%f (%f) with: %r" % (mean, stdev, param))	- Best model is printed.
#Output: Best: 0.975633 using {'activ': 'relu'} 0.974650 (0.001138) with: {'activ': 'softmax'} 0.975633 (0.002812) with: {'activ': 'relu'}	- Relu gave a slightly better accuracy on this dataset. - Keras is internally using Tensorflow

Other tools: Hyperopt, Hyperas(Variation of Hyperopt) – Hyperparameter tuning toolboxes which can work with keras and tensorflow.

- SKLearn is used due to prior experience in GridSearchCV and RandomizedSearchCV. Sufficient for most practical purposes.
- If you use CPU, take n_jobs = -1 gives faster results. For GPU, ignore n_jobs argument.

4. Summary of comments

a. It is softmax and relu the choices for activation function.

b. Keras comes with the new hyper parameter tuning library. It is better than keras.wrappers.
Please go through the video of sentdex and documentation

https://keras-team.github.io/keras-tuner/
https://github.com/keras-team/keras-tuner
https://www.youtube.com/watch?v=vvC15l4CY1Q

c. In classical ML we plotted the train acc. and cv acc. both to check overfitting of our model. While in GridSearchCV or Talos, we just get the parameters for which CV acc. is highest. Then, how are we sure that we're not overfitting?
I mean suppose I get a hyperparameter set (h1) from GridSearch with highest cv acc. = 86 % but the train acc. for h1 = 99%. Now, if another hyperparameter set (h2) exists such that cv acc. = 84% and train acc. = 87%.
Now, acc. to what we learnt in classical ML, h2 is a better set of hyperparameters as cv acc. of h2 ~ cv acc. of h1 but the difference between the train and cv acc. for h2 (3%) is much less than that for h1 (13%). Hence, h2 performs better generalization on unseen data. But, from clf.best_estimators() we're going to get h1 as it has the highest cv acc. How to deal with this problem?

Ans. You have to consider that hyperparameter which gives highest CV accuracy in cross validation. Whichever gives highest CV score, that will be the best

5. Additional material

a. https://github.com/autonomio/talos
b. We have one of our students build a case-study and write a blog on this exact topic. It's a two-part blog that you can read here:
Part I: https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-i-hyper-parameter-8129009f131b
Part II: https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7
c. About Hyperparameter Tuning in Keras:
https://towardsdatascience.com/hyperparameter-optimization-with-keras-b82e6364ca53
d. https://towardsdatascience.com/hyperparameter-optimization-with-keras-b82e6364ca53
e. https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

6. QAs

a. What can go wrong if you tune hyperparameters using the test set?
b. Can you list all hyperparameters you can tweak in a basic MLP? How could you tweak these hyperparameters to avoid overfitting?

53.1 BIOLOGICAL INSPIRATION: VISUAL CORTEX

Most popular for visual tasks: images; example: MNIST, Object Recognition...

In 1981, Nobel prize → Hubel and Wiesel → Research on Visual perception

They found that a certain neurons get fired to certain image features; in human brains certain visual areas have specialized functional properties.

- Some neuronsin the visual cortex fire when presented with line at specific orientation
- Different regions of the brain are responsible for edge-detection, motion, depth, color, shapes, faces
- There is hierarchical structure amongst these different regions, layers stacked

53.2 CONVOLUTION: EDGE DETECTION ON IMAGES.

Primary visual cortex performs Edge detection. Here we are going to understand CNN through edge detection.

Say we have a 6x6 matrix with 3 rows black and 3 rows white. Thus we have an edge at third-fourth row interface. Grayscale image pixel intensities - 0: black and 255: white.

0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
255	255	255	255	255	255
255	255	255	255	255	255
255	255	255	255	255	255

* Conv.

+1	+2	+1
0	0	0
-1	-2	-1

Sobel/edge detection (horizontal)

With convolutions, we have element wise multiplications and summation over them to get an element for a new matrix. The scope of the multiplications is determined by the size of the filter.

We will have a new matrix of size (**n-k+1, n-k+1**): n - size of input matrix, k – size of filter matrix

The above Sobel filter is used for detecting horizontal edges. Transpose of this matrix can be used to detect vertical edges.

Output Matrix of above convolution:

0	0	0	0
-1020	-1020	-1020	-1020
-1020	-1020	-1020	-1020
0	0	0	0

Normalization of this matrix: Min-max

255	255	255	255
0	0	0	0
0	0	0	0
255	255	255	255

Link: https://en.wikipedia.org/wiki/Sobel_operator

Such distinguished filters are applied to get different features of images for visual tasks in CNN.

Machine Learning: dot products on vectors

Convolution: Element wise multiplications and addition, dot products on matrices (generally)

Convolution can be applied on vectors also

53.3 CONVOLUTION: PADDING AND STRIDES

In previous topic, we had an input image of size 6x6 and the output array is of size 4x4. But if we want to have output array of a different size, we should go for padding which will add a row and column, top, bottom, left and right. But zero padding will generate extra edges. We can have same value padding. With padding of size p, we will have final matrix of size **(n-k+2p+1, n-k+2p+1)**.

Strides will help us skip rows and columns by a value equal to strides (s). We will have an output matrix of size (**int((n-k)/s) + 1, int((n-k)/s) + 1**)

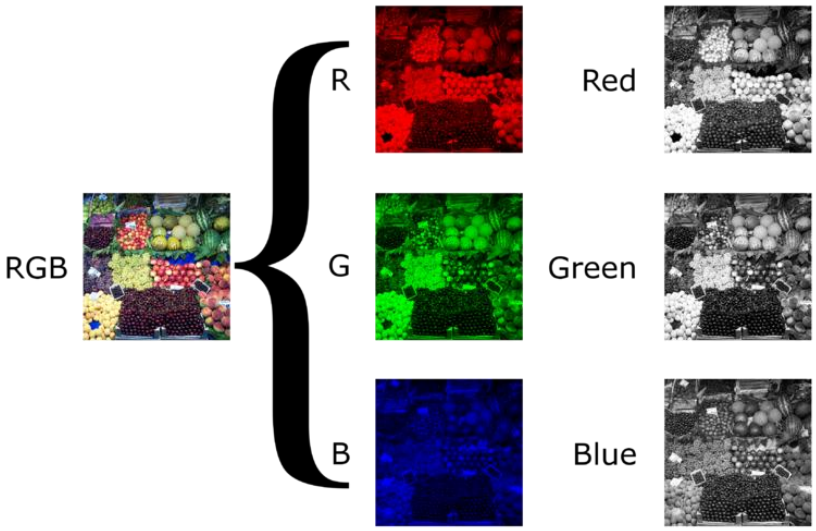
Convolution: element wise multiplication and addition

Padding: add rows and columns to get a desired output size

Strides: reduce size by a large factor

53.4 CONVOLUTION OVER RGB IMAGES

Each pixel in an RGB image contains a tuple of three values



It can also be thought of having three images stacked one over the other; resulting in a 3D Tensor; these multiple images are called as channels, So each image will have n x m x c size where c is the number of channels; Convolution filter will be a 3D tensor; care should be taken for image processing that the number of channels in filter will be same as the number of channels in the image; Convolution on a 3D image (n x n) with a 3D filter (k x k) results in a 2D array (output image) of size (n-k+1, n-k+1)

53.5 CONVOLUTIONAL LAYER

Link: http://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf, http://cs231n.github.io/convolutional-networks/#pool

We learnt: Convolution, Padding and Strides

Recap: In MLP, we have weights; input and bias values, the dot product of these values are filtered through an activation function

- Convolution layer – biologically inspired to process visual tasks
- Multiple edge detectors: multiple kernels
- In CNN we train the models to learn kernel matrices by back-propagation
- At each convolution layer we will have multiple kernels to learn different features of the images. For each kernel we will have a 2D array output, multiple kernels result in multiple output arrays (padded to get input array size), so at the layer we will get an output array of size n x n x m, where m is number of kernels (m is a hyper parameter)
- For every element in the output of filtering, the activation function is applied
- Pad, convolve and activate to transform an input array to an output array in a convolution layer
- Multiple layers of convolutions are used, at each layer we will extract features:
    - In image recognition: Pixels →edge →….. → part → object
    - Text: character →word → word group → sentence → story
    - Speech → sample → spectral brand → sound → … → phoneme → word
- MLP and convolution layers have similarity in terms of weights: kernels, while we train the models to learn weights in MLPs, we train the models to learn kernels in Conv Nets

53.6 MAX-POOLING.

Pooling introduces: Introduces small amount of Location invariance, Scale invariance and Rotational invariance

Pooling subsamples input arrays to reduce computational load, memory usage and number of parameters (limiting the risk of overfitting)

Pooling is destructive; sometimes invariance is not desirable,

We can also have a goal of equivariance, where a small change in input array should also reciprocate a small change in output (invariance: a change in input image does not show changes in output array)

Global Average Pooling: computes mean of entire map; gives out a single scalar

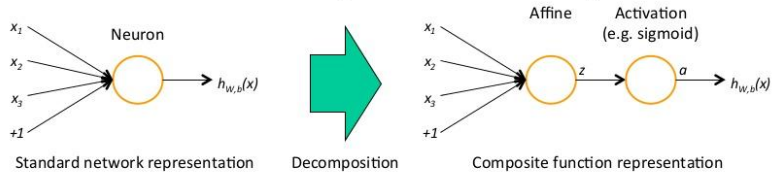
Additional Material: https://www.slideshare.net/kuwajima/cnnbp

Neural Network as a Composite Function

A neural network is decomposed into a composite function where each function element corresponds to a differentiable operation.

- Single neuron (the simplest neural network) example

A single neuron is decomposed into a composite function of an affine function element parameterized by W and b and an activation function element f which we choose to be the sigmoid function.



$$h_{W,b}(x) = f\left(W^T x + b\right) = \text{sigmoid}\left(\text{affine}_{W,b}(x)\right) = \left(\text{sigmoid} \circ \text{affine}_{W,b}\right)(x)$$

Derivatives of both affine and sigmoid function elements w.r.t. both inputs and parameters are known.
Note that sigmoid function does not have neither parameters nor derivatives parameters.

Sigmoid function is applied element-wise. '$\bullet$' denotes Hadamard product, or element-wise product.

$$\frac{\partial a}{\partial z} = a \bullet (1 - a) \text{ where } a = h_{w,b}(x) = \text{sigmoid}(z) = \frac{1}{1 + \exp(-z)}$$

$$\frac{\partial z}{\partial x} = W,\ \frac{\partial z}{\partial W} = x,\ \frac{\partial z}{\partial b} = I \text{ where } z = \text{affine}_{W,b}(x) = W^T x + b, \text{ and } I \text{ is identity matrix}$$

Chain Rule of Error Signals and Gradients

Error signals are defined as the derivatives of any cost function *J* which we choose to be the square error. Error signals are computed (propagated backward) by the chain rule of derivative and useful for computing the gradient of the cost function.

- Single neuron example

  Suppose we have *m* labeled training examples $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$. Square error cost function for each example is as follows. Overall cost function is the summation of cost functions over all examples.

$$J(W,b;x,y) = \frac{1}{2}\left\| y - h_{w,b}(x) \right\|^2$$

  Error signals of the square error cost function for each example are propagated using derivatives of function elements w.r.t. inputs.

$$\delta^{(a)} = \frac{\partial}{\partial a} J(W,b;x,y) = -(y-a)$$

$$\delta^{(z)} = \frac{\partial}{\partial z} J(W,b;x,y) = \frac{\partial J}{\partial a}\frac{\partial a}{\partial z} = \delta^{(a)} \bullet a \bullet (1-a)$$

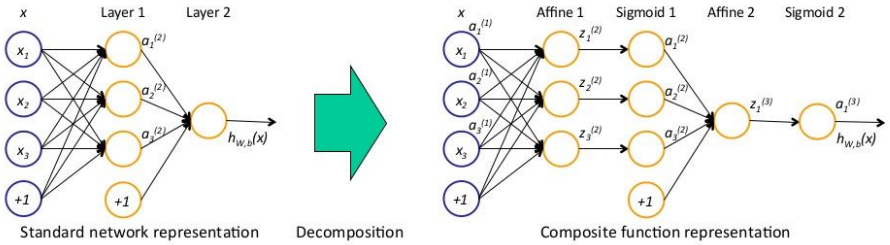
  Gradient of the cost function w.r.t. parameters for each example is computed using error signals and derivatives of function elements w.r.t. parameters. Summing gradients for all examples gets overall gradient.

$$\nabla_W J(W,b;x,y) = \frac{\partial}{\partial W} J(W,b;x,y) = \frac{\partial J}{\partial z}\frac{\partial z}{\partial W} = \delta^{(z)} x^T$$

$$\nabla_b J(W,b;x,y) = \frac{\partial}{\partial b} J(W,b;x,y) = \frac{\partial J}{\partial z}\frac{\partial z}{\partial b} = \delta^{(z)}$$

Decomposition of Multi-Layer Neural Network

- Composite function representation of a multi-layer neural network



$$h_{W,b}(x) = \left(\text{sigmoid} \circ \text{affine}_{W^{(2)},b^{(2)}} \circ \text{sigmoid} \circ \text{affine}_{W^{(1)},b^{(1)}}\right)(x)$$

- Derivatives of function elements w.r.t. inputs and parameters

$$a^{(1)} = x, a^{(l_{\max})} = h_{w,b}(x)$$

$$\frac{\partial a^{(l+1)}}{\partial z^{(l+1)}} = a^{(l+1)} \bullet \left(1 - a^{(l+1)}\right) \text{ where } a^{(l+1)} = \text{sigmoid}\left(z^{(l+1)}\right) = \frac{1}{1+\exp\left(-z^{(l+1)}\right)}$$

$$\frac{\partial z^{(l+1)}}{\partial a^{(l)}} = W^{(l)}, \frac{\partial z^{(l+1)}}{\partial W^{(l)}} = a^{(l)}, \frac{\partial z^{(l+1)}}{\partial b^{(l)}} = I \text{ where } z^{(l+1)} = \left(W^{(l)}\right)^T a^{(l)} + b^{(l)}$$

Error Signals and Gradients in Multi-Layer NN

- Error signals of the square error cost function for each example

$$\delta^{\left(a^{(l)}\right)} = \frac{\partial}{\partial a^{(l)}} J\left(W,b;x,y\right) = \begin{cases} -\left(y - a^{(l)}\right) \text{ for } l = l_{\max} \\ \dfrac{\partial J}{\partial z^{(l+1)}} \dfrac{\partial z^{(l+1)}}{\partial a^{(l)}} = \left(W^{(l)}\right)^T \delta^{\left(z^{(l+1)}\right)} \text{ otherwise} \end{cases}$$

$$\delta^{\left(z^{(l)}\right)} = \frac{\partial}{\partial z^{(l)}} J\left(W,b;x,y\right) = \frac{\partial J}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} = \delta^{\left(a^{(l)}\right)} \bullet a^{(l)} \bullet \left(\mathbf{1} - a^{(l)}\right)$$

- Gradient of the cost function w.r.t. parameters for each example

$$\nabla_{W^{(l)}} J\left(W,b;x,y\right) = \frac{\partial}{\partial W^{(l)}} J\left(W,b;x,y\right) = \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial W^{(l)}} = \delta^{\left(z^{(l+1)}\right)} \left(a^{(l)}\right)^T$$

$$\nabla_{b^{(l)}} J\left(W,b;x,y\right) = \frac{\partial}{\partial b^{(l)}} J\left(W,b;x,y\right) = \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial b^{(l)}} = \delta^{\left(z^{(l+1)}\right)}$$

Backpropagation in General Cases

1. Decompose operations in layers of a neural network into function elements whose derivatives w.r.t inputs are known by symbolic computation.

   $h_\theta(x) = \left(f^{(l_{\max})} \circ \ldots \circ f^{(l)}_{\theta^{(l)}} \circ \ldots \circ f^{(2)}_{\theta^{(2)}} \circ f^{(1)}\right)(x)$ where $f^{(1)} = x$, $f^{(l_{\max})} = h_\theta(x)$ and $\forall l : \dfrac{\partial f^{(l+1)}}{\partial f^{(l)}}$ is known

2. Backpropagate error signals corresponding to a differentiable cost function by numerical computation (Starting from cost function, plug in error signals backward).

$$\delta^{(l)} = \frac{\partial}{\partial f^{(l)}} J\left(\theta;x,y\right) = \frac{\partial J}{\partial f^{(l+1)}} \frac{\partial f^{(l+1)}}{\partial f^{(l)}} = \delta^{(l+1)} \frac{\partial f^{(l+1)}}{\partial f^{(l)}} \text{ where } \frac{\partial J}{\partial f^{(l_{\max})}} \text{ is known}$$

3. Use backpropagated error signals to compute gradients w.r.t. parameters only for the function elements with parameters where their derivatives w.r.t parameters are known by symbolic computation.

$$\nabla_{\theta^{(l)}} J\left(\theta;x,y\right) = \frac{\partial}{\partial \theta^{(l)}} J\left(\theta;x,y\right) = \frac{\partial J}{\partial f^{(l)}} \frac{\partial f^{(l)}_{\theta^{(l)}}}{\partial \theta^{(l)}} = \delta^{(l)} \frac{\partial f^{(l)}_{\theta^{(l)}}}{\partial \theta^{(l)}} \text{ where } \frac{\partial f^{(l)}_{\theta^{(l)}}}{\partial \theta^{(l)}} \text{ is known}$$

4. Sum gradients over all example to get overall gradient. $\nabla_{\theta^{(l)}} J\left(\theta\right) = \sum_{i=1}^{m} \nabla_{\theta^{(l)}} J\left(\theta;x^{(i)},y^{(i)}\right)$

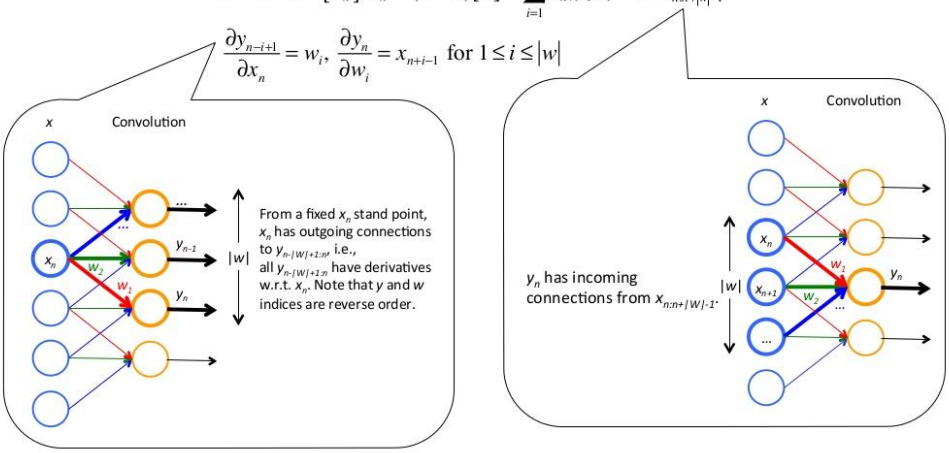
Derivatives of Convolution

- Discrete convolution parameterized by a feature *w* and its derivatives

Let *x* be the input, and *y* be the output of convolution layer. Here we focus on only one feature vector *w*, although a convolution layer usually has multiple features $W = [w_1\ w_2\ \dots\ w_n]$. *n* indexes *x* and *y* where $1 \le n \le |x|$ for x_n, $1 \le n \le |y| = |x| - |w| + 1$ for y_n. *i* indexes *w* where $1 \le i \le |w|$. $(f*g)[n]$ denotes the *n*-th element of $f*g$.

$$y = x * w = [y_n], y_n = (x * w)[n] = \sum_{i=1}^{|w|} x_{n+i-1} w_i = w^T x_{n:n+|w|-1}$$

$$\frac{\partial y_{n-i+1}}{\partial x_n} = w_i,\ \frac{\partial y_n}{\partial w_i} = x_{n+i-1} \text{ for } 1 \le i \le |w|$$



Backpropagation in Convolution Layer

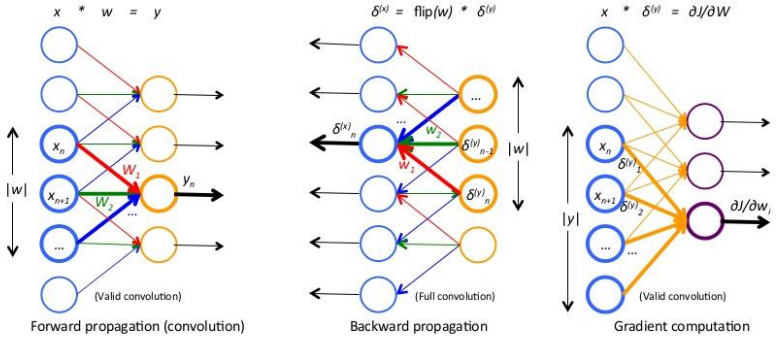
Error signals and gradient for each example are computed by convolution using the commutativity property of convolution and the multivariable chain rule of derivative.

Let's focus on single elements of error signals and a gradient w.r.t. *w*.

$$\delta_n^{(x)} = \frac{\partial J}{\partial x_n} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial x_n} = \sum_{i=1}^{|w|}\frac{\partial J}{\partial y_{n-i+1}}\frac{\partial y_{n-i+1}}{\partial x_n} = \sum_{i=1}^{|w|}\delta_{n-i+1}^{(y)} w_i = \left(\delta^{(y)} * \text{flip}(w)\right)[n],\ \delta^{(x)} = \left[\delta_n^{(x)}\right] = \delta^{(y)} * \text{flip}(w)$$

↑ Reverse order linear combination

$$\frac{\partial J}{\partial w_i} = \frac{\partial J}{\partial y}\frac{\partial y}{\partial w_i} = \sum_{n=1}^{|x|-|w|+1}\frac{\partial J}{\partial y_n}\frac{\partial y_n}{\partial w_i} = \sum_{n=1}^{|x|-|w|+1}\delta_n^{(y)} x_{n+i-1} = \left(\delta^{(y)} * x\right)[i],\ \frac{\partial J}{\partial w} = \left[\frac{\partial J}{\partial w_i}\right] = \delta^{(y)} * x = x * \delta^{(y)}$$



Derivatives of Pooling

Pooling layer subsamples statistics to obtain summary statistics with any aggregate function (or filter) *g* whose input is vector, and output is scalar. Subsampling is an operation like convolution, however *g* is applied to disjoint (non-overlapping) regions.

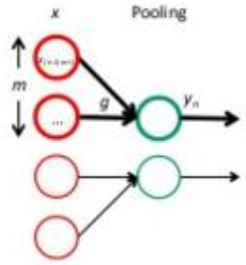
- Definition: *subsample* (or *downsample*)

  Let *m* be the size of pooling region, *x* be the input, and *y* be the output of the pooling layer. subsample(*f*, *g*)[*n*] denotes the *n*-th element of subsample(*f*, *g*).

$$y_n = \text{subsample}(x, g)[n] = g\left(x_{(n-1)m+1:nm}\right)$$

$$y = \text{subsample}(x, g) = [y_n]$$

$$g(x) = \begin{cases} \dfrac{\sum_{k=1}^{m} x_k}{m}, \dfrac{\partial g}{\partial x} = \dfrac{\mathbf{1}}{m} & \text{mean pooling} \\ \max(x), \dfrac{\partial g}{\partial x_j} = \begin{cases} 1 \text{ if } x_j = \max(x) \\ 0 \text{ otherwise} \end{cases} & \text{max pooling} \\ \|x\|_p = \left(\sum_{k=1}^{m} |x_k|^p\right)^{1/p}, \dfrac{\partial g}{\partial x_i} = \left(\sum_{k=1}^{m} |x_k|^p\right)^{1/p-1} |x_i|^{p-1} & \text{L}^p \text{ pooling} \\ \text{or any other differentiable } \mathbf{R}^m \to \mathbf{R} \text{ functions} \end{cases}$$



Backpropagation in Pooling Layer

Error signals for each example are computed by upsampling. Upsampling is an operation which backpropagates (distributes) the error signals over the aggregate function *g* using its derivatives $g'_n = \partial g / \partial x_{(n-1)m+1:nm}$. g'_n can change depending on pooling region *n*.

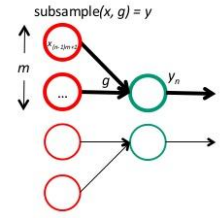
- In max pooling, the unit which was the max at forward propagation receives all the error at backward propagation and the unit is different depending on the region *n*.

- Definition: *upsample*

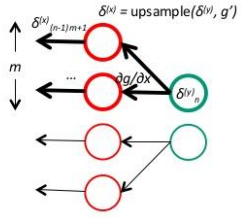
  upsample(*f*, *g*)[*n*] denotes the *n*-th element of upsample(*f*, *g*).

$$\delta^{(x)}_{(n-1)m+1:nm} = \text{upsample}\left(\delta^{(y)}, g'\right)[n] = \delta^{(y)}_n g'_n = \delta^{(y)}_n \frac{\partial g}{\partial x_{(n-1)m+1:nm}} = \frac{\partial J}{\partial y_n} \frac{\partial y_n}{\partial x_{(n-1)m+1:nm}} = \frac{\partial J}{\partial x_{(n-1)m+1:nm}}$$

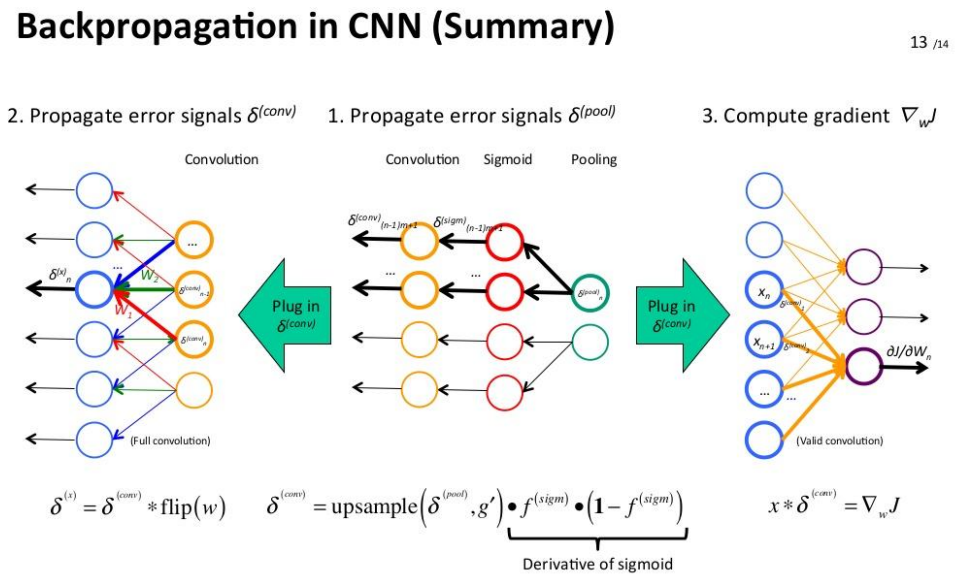
$$\delta^{(x)} = \text{upsample}\left(\delta^{(a)}, g'\right) = \left[\delta^{(x)}_{(n-1)m+1:nm}\right]$$



Forward propagation (subsapmling)



Backward propagation (upsapmling)



53.7 CNN TRAINING: OPTIMIZATION

In MLPs, back propagation can be applied;

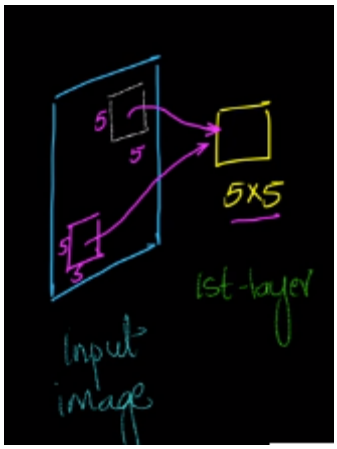
Convolution layer: Similar to MLP - del z/ del W = x

Max pooling back propagation: gradient propagation to only the maximum value, say from a 2x2 matrix we max pool to get a 1x1 scalar, when back propagate through max pooling gradient in the 2x2 matrices are either 0 or 1, it 1 where the maximum is present and 0 everywhere else. Gradient = 1 because the same value is pulled as output (maximum valued element and non-max values have no effect on the output).

Optimization is similar to an MLP

53.8 RECEPTIVE FIELDS AND EFFECTIVE RECEPTIVE FIELDS

Receptive field: when passing the image through a filter at any time the pixels at which the filter is being applied is the receptive field at that instant.



Effective Receptive field: In a Deep CNN, at a deeper layer a filter focuses on the pixels of input image. This region of focus on which layers of convolutions are applied is an effective receptive field. In layer 2, the filter has receptive field in the output of first layer and effective receptive field in the input image.

53.9 EXAMPLE CNN: LENET [1998]

Link: https://world4jason.gitbooks.io/research-log/content/deepLearning/CNN/Model%20&%20ImgNet/lenet.html

Concepts of techniques are old but we didn't have datasets and compute power;

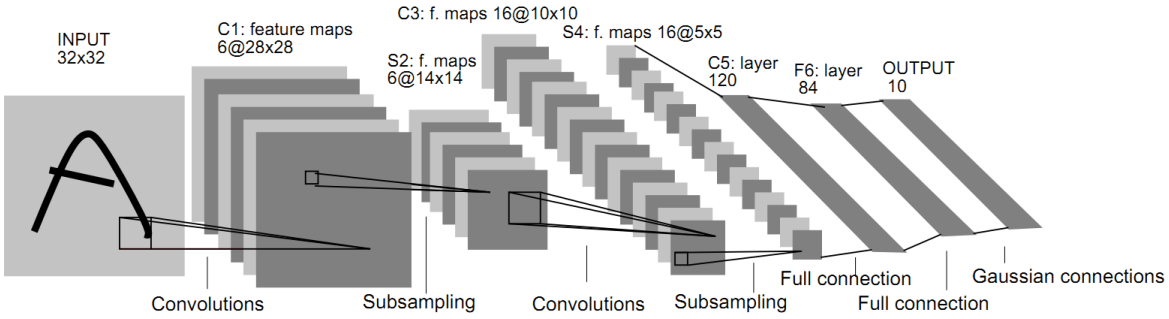


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet: Small depth

53.10 IMAGENET DATASET

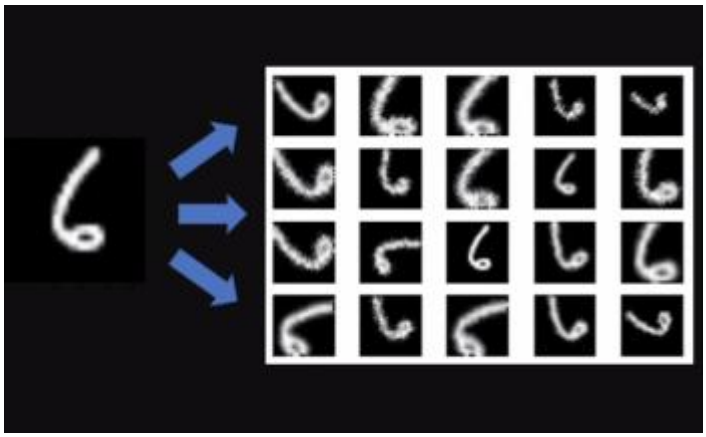
Link: https://en.wikipedia.org/wiki/ImageNet

Contributed the most to Deep Learning; This dataset became the benchmark dataset for all new DL algorithms

---

53.11 DATA AUGMENTATION

We want CNN models to be robust to input image changes, such as translation, scaling, mirror, etc. Thus, the input image is passed through data augmentation generating new images. For example: (using matrix transformation)



Link: https://github.com/albumentations-team/albumentations

Through data augmentation, a new dataset is created (through transformations)

Can introduce invariance in CNN, create a large dataset when we have small datasets.

53.12 CONVOLUTION LAYERS IN KERAS

Links: https://keras.io/layers/convolutional/

https://keras.io/layers/pooling/

https://keras.io/layers/core /

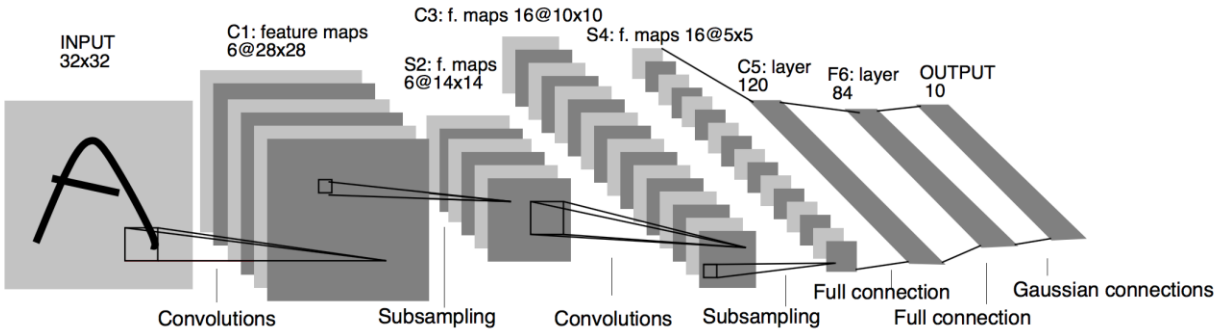
Flatten: Converts a 2D array into a 1D vector

53.13 ALEXNET

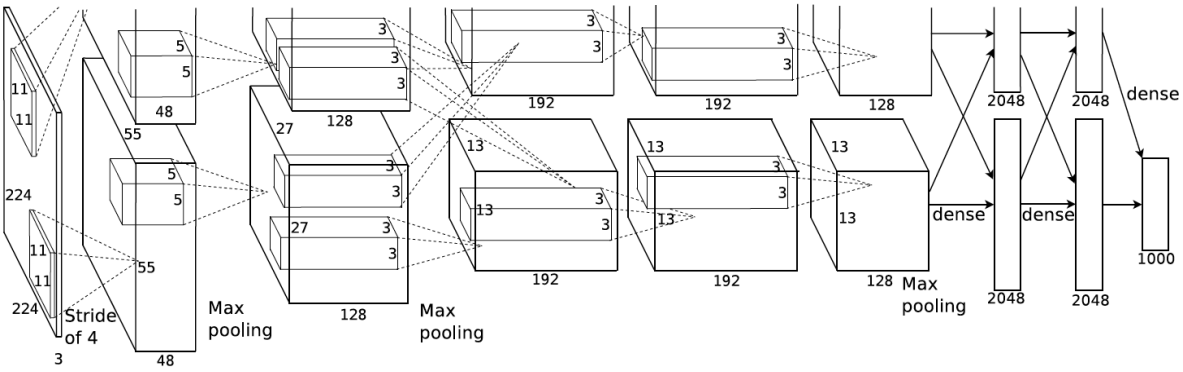
53.14 VGGNET

Developments in Deep Learning training on Imagenet dataset;

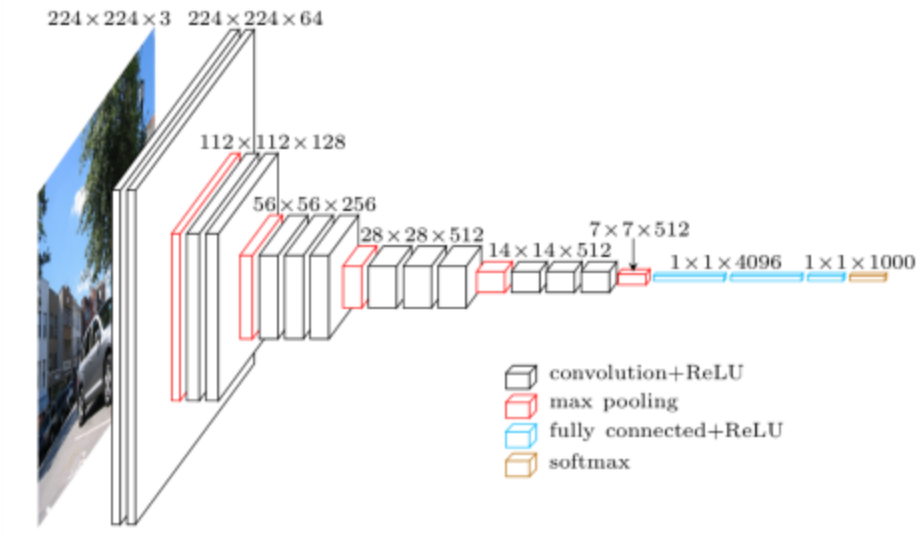
LeNet: 2 Conv, 2 Mean Pool, 3 FC – 1000s trainable params, Sigmoid Activation: First Architecture for Hand written images classification



AlexNet: 5 Conv, 3 Pool, 3 FC – 10M trainable params, ReLU activation, Includes Dropouts: Trained on ImageNet



VGGNet: (2Conv+1MaxPool)*2 + (3Conv+1MaxPool)*3 + 3FC + Softmax



Additional:

- The **Softmax classifier** gets its name from the **softmax** function, which is used to squash the raw class scores into normalized positive values that sum to one, so that the cross-entropy loss can be applied

---

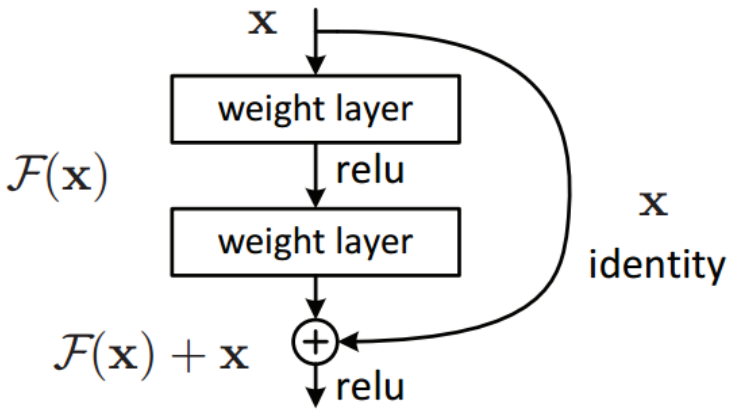
53.15 RESIDUAL NETWORK.

ResNets: Residual Networks: Regular networks as depth increases both training error and test error are increasing. This was tackled using ResNets. Skip connections: Output of previous layer is given as input to the next layer also in addition to giving it as input to the current layer.

ReLU(x) = x

So as increasing number of layers has an effect of increasing error, ResNets skip connection concept will ensure that some of the layers that are useless will be neglected (type of regularization). With ResNets we can add additional layers such that performance does not get effected. If the new layers are useful, performance will increase as skipping will not happen.

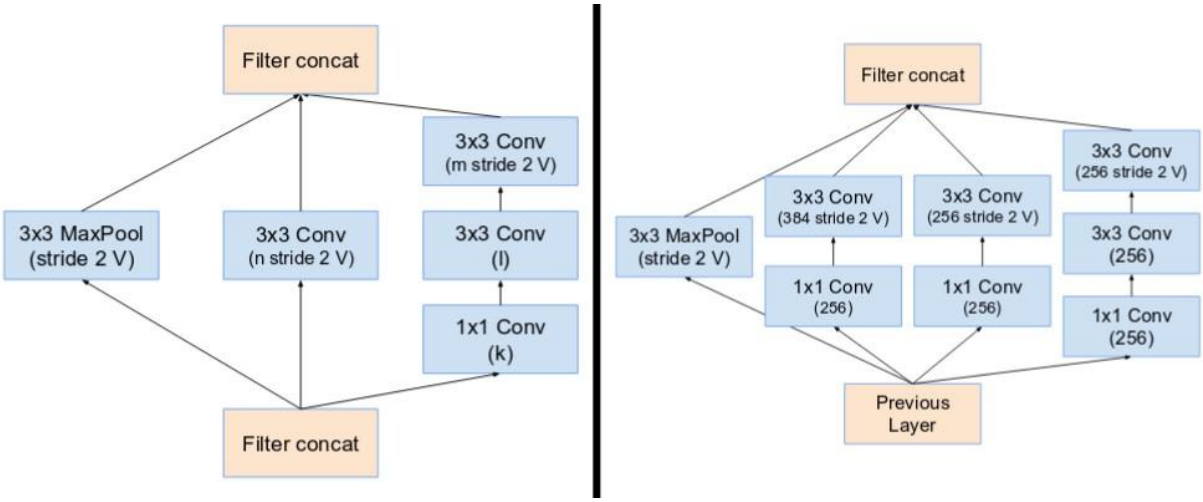
Input array dimensions should match while using skip connections. ResNets are used to avoid reduction of performance when number of layers increases.



53.16 INCEPTION NETWORK.

Inception Network: Using multiple filters at a layer and stacking output of each filter to result in a single output. This will help us take advantage of different filters or Kernel sizes. Number of computations is large for each filter. As we increase number of kernels at each layer we will have very large computations. It is of the order of Billion computations at each layer. Having an intermediate layer of size 1x1 before each filter will reduce the number of computations.

http://www.ashukumar27.io/CNN-Inception-Network/



53.17 WHAT IS TRANSFER LEARNING

Transfer Learning: Utilizing performance of an already trained neural network to work on a new dataset instead of building an NN from scratch to solve the Visual tasks. Pre-trained models are readily available on Keras and Tensorflow. The pre-trained model can be trained on one dataset and can be applied on to another related dataset. A model trained on cars can be applied to recognize trucks. Note that it takes a lot of time to train a model from scratch.

Link: http://cs231n.github.io/transfer-learning/

In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest. The three major Transfer Learning scenarios look as follows:

- **ConvNet as fixed feature extractor**. Take a ConvNet pretrained on ImageNet, remove the last fully-connected layer (this layer's outputs are the 1000 class scores for a different task like ImageNet), then treat the rest of the ConvNet as a fixed feature extractor for the new dataset. In an AlexNet, this would compute a 4096-D vector for every image that contains the activations of the hidden layer immediately before the classifier. We call these features **CNN codes**. It is important for performance that these codes are ReLUd (i.e. thresholded at zero) if they were also thresholded during the training of the ConvNet on ImageNet (as is usually the case). Once you extract the 4096-D codes for all images, train a linear classifier (e.g. Linear SVM or Softmax classifier) for the new dataset.
- **Fine-tuning the ConvNet**. The second strategy is to not only replace and retrain the classifier on top of the ConvNet on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the ConvNet, or it's

possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a ConvNet contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the ConvNet becomes progressively more specific to the details of the classes contained in the original dataset. In case of ImageNet for example, which contains many dog breeds, a significant portion of the representational power of the ConvNet may be devoted to features that are specific to differentiating between dog breeds.

- **Pretrained models**. Since modern ConvNets take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final ConvNet checkpoints for the benefit of others who can use the networks for fine-tuning. For example, the Caffe library has a Model Zoo where people share their network weights.

When and how to fine-tune? How do you decide what type of transfer learning you should perform on a new dataset? This is a function of several factors, but the two most important ones are the size of the new dataset (small or big), and its similarity to the original dataset (e.g. ImageNet-like in terms of the content of images and the classes, or very different, such as microscope images). Keeping in mind that ConvNet features are more generic in early layers and more original-dataset-specific in later layers, here are some common rules of thumb for navigating the 4 major scenarios:

1. *New dataset is small and similar to original dataset*. Since the data is small, it is not a good idea to fine-tune the ConvNet due to overfitting concerns. Since the data is similar to the original data, we expect higher-level features in the ConvNet to be relevant to this dataset as well. Hence, the best idea might be to train a linear classifier on the CNN codes.
2. *New dataset is large and similar to the original dataset*. Since we have more data, we can have more confidence that we won't overfit if we were to try to fine-tune through the full network.
3. *New dataset is small but very different from the original dataset*. Since the data is small, it is likely best to only train a linear classifier. Since the dataset is very different, it might not be best to train the classifier form the top of the network, which contains more dataset-specific features. Instead, it might work better to train the SVM classifier from activations somewhere earlier in the network.
4. *New dataset is large and very different from the original dataset*. Since the dataset is very large, we may expect that we can afford to train a ConvNet from scratch. However, in practice it is very often still beneficial to initialize with weights from a pretrained model. In this case, we would have enough data and confidence to fine-tune through the entire network.

Practical advice. There are a few additional things to keep in mind when performing Transfer Learning:

- *Constraints from pretrained models*. Note that if you wish to use a pretrained network, you may be slightly constrained in terms of the architecture you can use for your new dataset. For example, you can't arbitrarily take out Conv layers from the pretrained network. However, some changes are straight-forward: Due to parameter sharing, you can easily run a pretrained network on images of different spatial size. This is clearly evident in the case of Conv/Pool layers because their forward function is independent of the input volume spatial size (as long as the strides "fit"). In case of FC layers, this still holds true because FC layers can be converted to a Convolutional Layer: For example, in an AlexNet, the final pooling volume before the first FC layer is of size [6x6x512]. Therefore, the FC layer looking at this volume is equivalent to having a Convolutional Layer that has receptive field size 6x6, and is applied with padding of 0.
- *Learning rates*. It's common to use a smaller learning rate for ConvNet weights that are being fine-tuned, in comparison to the (randomly-initialized) weights for the new linear classifier that computes the class scores of your new dataset. This is because we expect that the ConvNet weights are

relatively good, so we don't wish to distort them too quickly and too much (especially while the new Linear Classifier above them is being trained from random initialization).

53.18 CODE EXAMPLE: CATS VS DOGS

---

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

---

53.19 CODE EXAMPLE: MNIST DATASET

IPYNB

53.20 Interview question: How to build a face recognition system from scratch?

From scratch: we need around 1000 images per person.

If dataset size is less, use pre-trained models with fine tuning on around 100 images per person

Collect data in various conditions → Data augmentation → Transfer learning → Use Categorical log loss (cross entropy with Softmax) →Try cloud APIs for faster training (Microsoft Azure, Google Images, etc.)

54.1 WHY RNNS?

Suited for sequences of data, of words, etc.; in most sentences the sequence becomes important in addition to presence of a word

In all vectorization methods of textual data which work on the occurrence of a word, they discard the semantic meaning or sequence information; Machine Translation: sequence of French sentence to English; Speech recognition

In time series data, at new instant we have an observation. (Stock market, ride pickups)

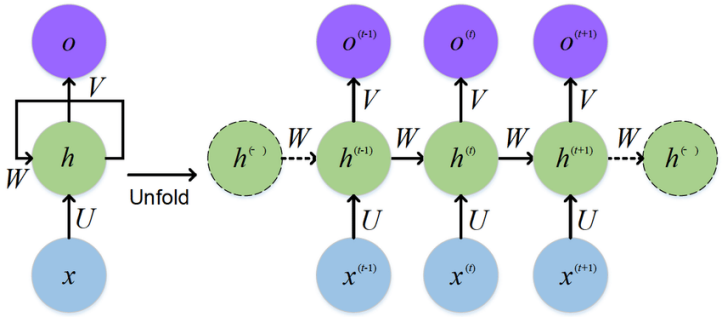
In image captioning, we have output as sequence

Also sentences can have different lengths; we can zero pad all sentences to a common length. But this is memory inefficient, high number of parameters.

RNNS: Need: each input is of different length, number of parameters should be less

54.2 RECURRENT NEURAL NETWORK

Recurrent: repeating



Ex:

Let us take a Task: Classify sequence of words into 0 or 1

X1 – (X11, X12, X13, X14, X15)

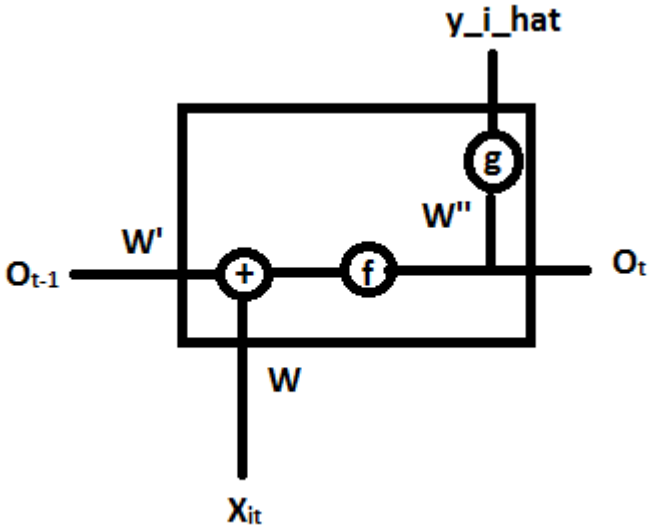
At t =1, X11 is given input to a layer, this gives an output O1 that is taken as input for the same layer at t = 2. O1 depends on input at t =1 that is on X11. O2 depends on input at t = 2, X12 and also on O1.

O1 = f(W*X11), O2 = f(W*X12 + W'*O1), O3 = f(W*X13 + W'*O2), …

Let O5 be the output of final time stamp. On O5 we will apply an activation function to get y_i_hat. This activation function will have a Weight Matrix.

Three weight matrices: W for input, W' for previous state output and W'' before the final activation function. To create this as a repetitive structure, we can have a dummy output (O0) before the first input. Weights can be initiated using Xavier/ Glorot method.

To get output we apply an activation function to the output of the desired time step cell.



→RNN

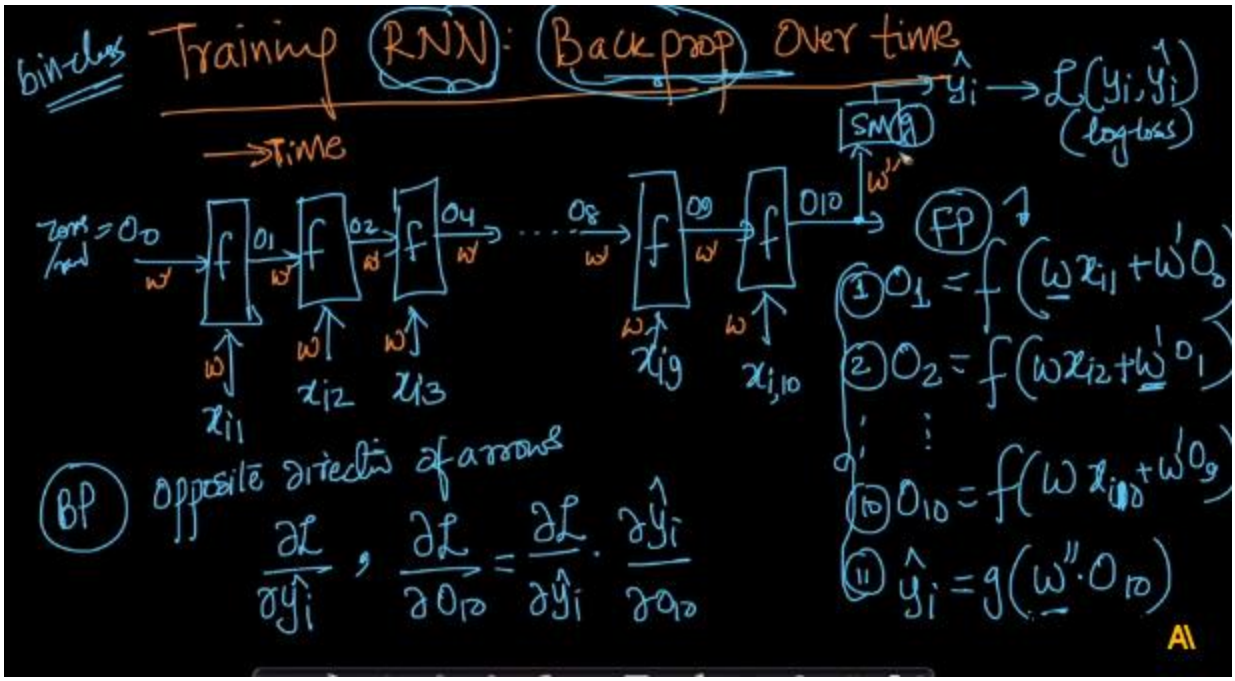
The same neurons are used over the whole sequence of words of the sentence, the RNN layer is generally confused to be of a number of layers at different time steps, which is wrong.

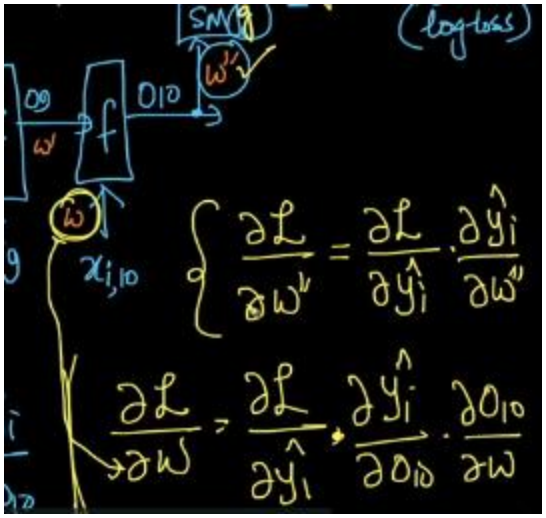
54.3 TRAINING RNNS: BACKPROP

For RNNs Back propagation through time is used. Back propagation is unrolled over time.

RNN: Only one layer repeating over time

For gradient computations follow the arrows





All the weights are same (there are only three weight matrices)

At the end of back propagation, multiplications are large which results in vanishing or exploding gradients. The number of layers may be low but the because of recurrence of the computations which depends on the length of sequences.

So I got the hang of these RNN Back prop through time concept: for cases such as one to many or many to many. It is just that we calculate del l/ del w at each time step and update W. While calculating the gradient we follow all the paths from Loss function to the weight matrices. For ex. Del l / del w = **(** del l / del y2_hat * del y2_hat/ del On * del On/ del O(n-1) … * del Ok/ del W **)** + **(** del l / del y1_hat * del y1_hat/ del O(n-1) * del O(n-1)/ del O(n-2) … * del Ok/ del W **)** + other outputs on which Loss function is depended on.

54.4 TYPES OF RNNS

Many to one RNN (sentiment classification);

Many to many: same length: Parts of speech,

Many to many: different length: Machine Translation;

One to many: Image captioning (Input: Image matrix, Output: Sequence of words)

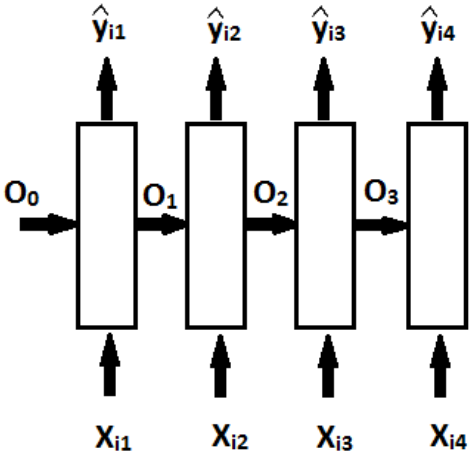
One to one: MLP

54.5. NEED FOR LSTM/GRU.

Video Lecture

What is the problem with Simple RNNs?

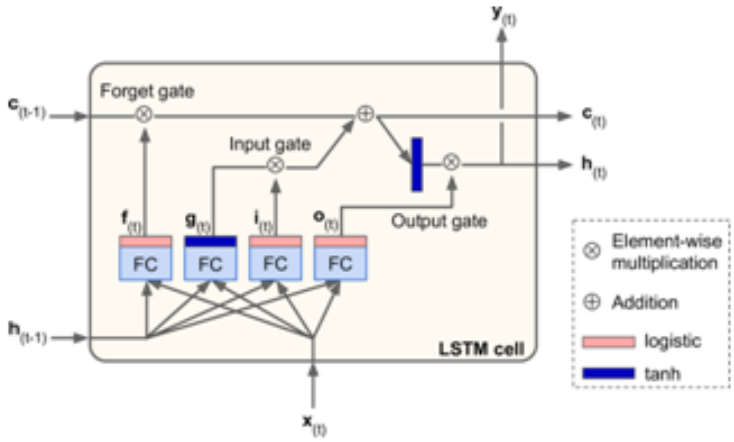
Many to many same length RNN:



y_{i4} depends a lot on x_{i4} and O_3 and depends less on x_{i1} and O_1 due to vanishing gradients

Simple RNNs **cannot take care of long term dependency**, when 4^{th} output depends a lot on 1^{st} input

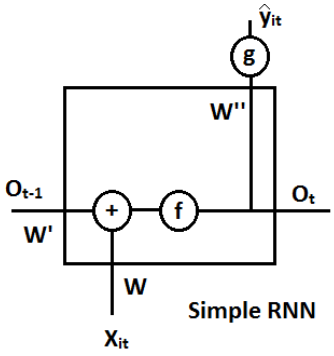
54.6 LSTM.



Lecture:

LSTM: takes care of long term dependencies as well as short term dependencies

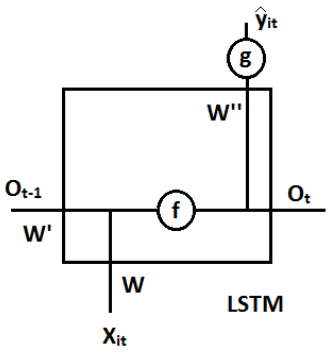
LSTM: Long Short Term Memory



Simple RNN:

$O_t = f(Wx_{it} + W'O_{t-1})$ (sum)

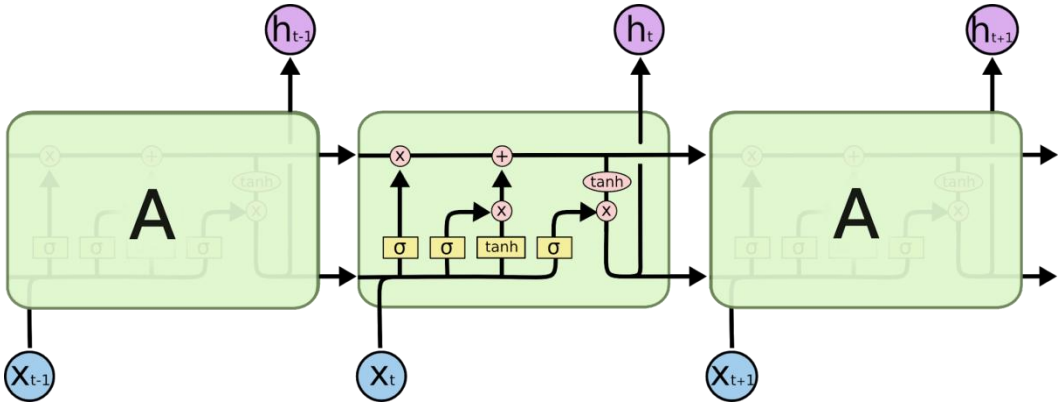
y_{it} hat $= g(W'' O_t)$



LSTM:

$O_t = f([W,W''] [x_{it}\ O_{t-1}])$ - concatenation

y_{it} hat = $g(W''\ O_t)$



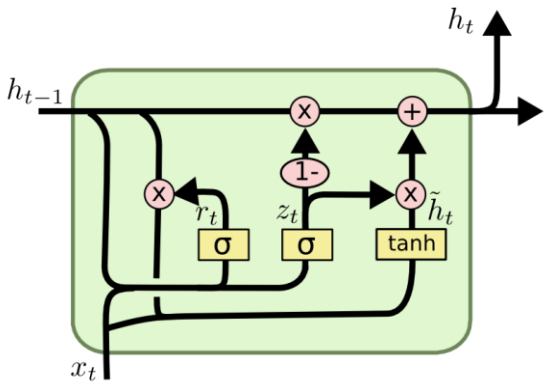
- 3 inputs: previous cell state, input, output of previous cell
- 2 outputs: current output, current cell state
- Using identity we can pass same cell states
- Amount of previous cell state you want to pass can be manipulated

Additional material

a. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

54.7 GRU

Detailed explanation



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-var-GRU.png

- GRU cell is a simplified version of the LSTM cell and it performs as well as LSTM
- Cell states from previous cell are merged into a single vector
- Gate controller controls the forget gate and the input gate
- There is no separate output gate, the full cell state is given as output
- Usage: keras.layers.GRU()
- Still it is difficult to retain very-long term patterns with these RNNs

Video Lecture

- GRU: Gated Recurrent Unit
- LSTMS were created in 1997, GRUs - 2014
- LSTM have 3 gates: input, output, forget
- GRUs: simplified version inspired from LSTMs, faster to train, as powerful as LSTM
- GRUs: 2 gates: reset, update
- The short-circuit structure is necessary for having long term dependencies

Summary of comments

Applications of long term dependencies: https://en.wikipedia.org/wiki/Long_short-term_memory#Applications

- Predicting sales, finding stock market trends
- Understanding movie plots
- Speech recognition
- Music composition

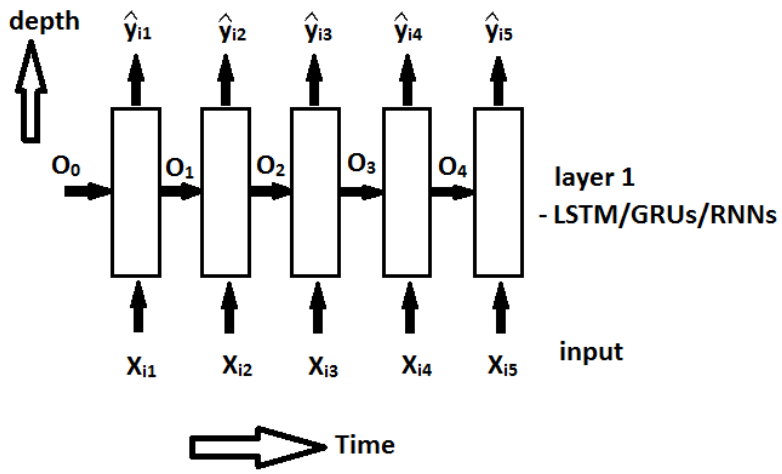
Additional material

a. https://www.slideshare.net/hytae/recent-progress-in-rnn-and-nlp-63762080
b. https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be
c. https://datascience.stackexchange.com/questions/14581/when-to-use-gru-over-lstm

54.8 DEEP RNN.

Video Lecture

- In MLPs, built one layer and then extended it to Deep MLPs
- In CNN, built one layer and then extended it to Deep CNN
- Similarly, with GRUs/LSTMs we will extend it to Deep RNNs



- Stacking one layer one over the other to build multiple layers
- Deep RNN structure as other NNs are problem specific
- Backpropagation and Forward propagation can be determined by the direction of arrows
- Back propagation occurs across time and across depth also

Summary of comments

- The number of units in each layer of RNN is a hyperparameter that we need to tune. They're not dependent on the number of words in a sentence.
- Number of words in sentence == Number of time a cell will be unfolded along time axis == Maximum length of sentence among all sentences != number of units
- Seems like you are getting confused between the embedding and padding. Please note that padding is different from embedding. We make padding to sentences whereas here embedding is done to words. Suppose we have a sentence with 8 words and we pad them to 10 by adding two zeros in the end, we now have a 1*10 vector. Now using embedding layer, if we want to represent each word using some 5 dimensional matrix, then we will have 1*10*5 . At each

time step we send in 1*5 vector and we send 10 such vectors to the LSTM unit. We can send sentences without padding as well. But we can send only one sentence at a time if we don't use padding. But we cannot send two words of different dimensions into LSTM as the length of cell state, hidden state are fixed.

- Forget gate parameters are learnt by backpropagation.
- Memoization is taken care by Tensorflow backend
- During Backprop, weights are updated after all time steps.

Additional material

a. https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/

54.9 BIDIRECTIONAL RNN.

Detailed explanation

In a regular RNN, at each time step the layer just looks at past and present inputs. In Machine Translation tasks it is important to look ahead in the words that come next. This is implemented by using two recurrent layers which start from the two extremities of the sentence, and combine their outputs at each time step.

Usage: keras.layers.Bidirectional(keras.layers.GRU())

Video Lecture

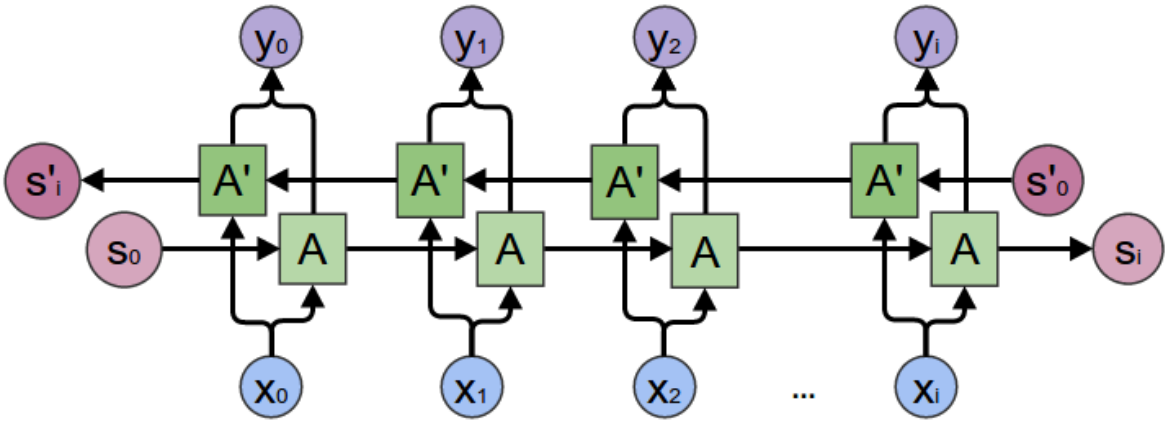
Bi- directional RNNs:

NLP: input: x, output: y

In simple RNNs, y_{i3} is assumed to be dependent on x_{11}, x_{12}, x_{13}

But if y_{i3} is dependent on x_{14}, x_{15} dependent which comes after y_{i3}'s time step.

To cater this we have bi-directional RNNs.



54.10 CODE EXAMPLE: IMDB SENTIMENT CLASSIFICATION

Video Lecture

- Dataset: IMDB reviews
  - LSTMs are tricky to get them to work.
  - Dataset: xi, yi pairs, xi - sequence of words:
    - xi is rank encoded list of the sentence, when you use top_words = k, the dataset will be loaded with k most frequent words
    - Each xi is represented with a list of indices of the top_words.
    - All xi are padded to length 600 (a common length for sequences). Pre-padded to have sequence at the last of time steps. If post-padded, the sequence will be far from the output time step and leads to the requirement of a strong Long-term memory layer.
    - For sequences with length >this padded length, the sequence is truncated
    - Without padding: SGD operation will be with batch_size = 1, which will be very slow. Sequences with different lengths cannot be formed into a batch as some xi require more time steps. To make batch type training possible, we use padding, else the training will be slow.
    - Embedding: Turns positive integers {indexes} into dense vectors of fixed size, can only be used as first layer in a model
    - Embed layer parameters: 5000*32 = 160k
    - LSTM: Number of parameters: (4*100(100+32+1))

        - Dense layer: 1 output: 1*(100+1) parameters
        - Each LSTM cell can learn different aspects of the sequence leading to having different weight matrices and each cell is unrolled over time.

Additional material

- Derivation of parameters in an LSTM layer (https://www.youtube.com/watch?v=l5TAISVlYM0&feature=youtu.be)

- LSTM layer with m units:
    - There are four gates:
        - “f” First gate: Bias, input n-dimensional vector, cell state m-dimensional, Weights vector m+n dimensional: Number of parameters: (n+m+1)
        - “i” Second gate: similar number of params
        - “c” Third gate: similar
        - “o” Fourth gate: similar
        - Parameters: 4(n+m+1) per cell
        - Total: 4m(n+m+1) per LSTM layer with m units, n = input size
        - = 4(nm+m**2 + m)

55

DEEP LEARNING: GENERATIVE ADVERSARIAL NETWORKS (GANs)

55.1 LIVE SESSION ON GENERATIVE ADVERSARIAL NETWORKS

56

LIVE: ENCODER-DECODER MODELS

56.1 LIVE: ENCODER-DECODER MODELS

57

ATTENTION MODELS IN DEEP LEARNING

57.1 ATTENTION MODELS IN DEEP LEARNING

--- Live Sessions notebook

58.1 QUESTIONS AND ANSWERS

Basics of NLP:

1. Explain about Bag of Words?
2. Explain about Text Preprocessing: Stemming, Stop-word removal, Tokenization, Lemmatization.
3. Explain about uni-gram, bi-gram, n-grams.?
4. What is tf-idf (term frequency- inverse document frequency)
5. Why use log in IDF?
6. Explain about Word2Vec.?
7. Explain about Avg-Word2Vec, tf-idf weighted Word2Vec?
8. Explain about Multi-Layered Perceptron (MLP)?
9. How to train a single-neuron model?
10. How to Train an MLP using Chain rule ?
11. How to Train an MLP using Memoization?
12. Explain about Backpropagation algorithm?
13. Describe about Vanishing and Exploding Gradient problem?
14. Explain about Bias-Variance tradeoff in neural Networks?

Deep Learning:

1. What is sampled softmax?
2. Why is it difficult to train a RNN with SGD?
3. How do you tackle the problem of exploding gradients? (By gradient clipping)
4. What is the problem of vanishing gradients? (RNN doesn't tend to remember much things from the past)
5. How do you tackle the problem of vanishing gradients? (By using LSTM)
6. Explain the memory cell of a LSTM. (LSTM allows forgetting of data and using long memory when appropriate.)
7. What type of regularization do one use in LSTM?
8. What is the problem with sigmoid during back propagation? (Very small, between 0.25 and zero.)
9. What is transfer learning?
10. What is back propagation through time? (BPTT)
11. What is the difference between LSTM and GRU?
12. Explain Gradient Clipping.
13. Adam and RMSProp adjust the size of gradients based on previously seen gradients. Do they inherently perform gradient clipping? If no, why?

Additional Source: https://www.analyticsvidhya.com/blog/2017/01/must-know-questions-deep-learning/